DOCUMENT RESUME

ED 348 971 IR 015 648

AUTHOR Norris, Cathleen A., Ed.; Poirot, James L. Ed.
TITLE Problem Solving and Critical Thinking for Computer Science Educators.
INSTITUTION International Society for Technology in Education, Eugene, OR.
REPORT NO ISBN-1-56484-007-7
PUB DATE 91
NOTE 67p.
PUB TYPE Viewpoints (Opinion/Position Papers, Essays, etc.) (120) -- Reports - Research/Technical (143)

EDRS PRICE MF01 Plus Postage. PC Not Available from EDRS.
DESCRIPTORS *Computer Assisted Instruction; Computer Software Development; *Critical Thinking; Educational Research; Elementary Secondary Education; Expert Systems; *Problem Solving; Programed Tutoring; Programing
IDENTIFIERS Knowledge Bases; *LOGO Programing Language

ABSTRACT

The eight papers presented in this monograph are a result of the Problem Solving and Critical Thinking Research Workshop that was held in conjunction with the 1990 National Educational Computing Conference (NECC). The intent of the workshop was to provide a unique forum for researchers to share ideas in a special area of educational computing. The monograph provides an overview of the general issues of problem solving and critical thinking in education as well as specialized areas of interest in intelligent tutoring and program construction. The papers included in this monograph are: (1) "Problem Solving, Critical Thinking, and Computing: An Overview" (Cathleen A. Norris and James L. Poirot); (2) "'Mindstorms' Revisited: Computers, Problem Solving, and Knowledge-based Instruction" (Karen Swan); (3) "Defining Programming and Logo as Vehicles for Developing Higher Order Thinking Skills" (Jim Dunne); (4) "Abstracted Knowledge: A Mid-Road Transfer Approach to Critical Thinking" (Clifton S. Harris); (5) "Resolving the Impasse in Software Engineering: Problem Solving in Program Construction" (Warren Moseley); (6) "Critical Thinking and Intelligent Tutoring Systems" (James T. Streib); (7) "Critical Thinking and Open Courseware" (Eduardo Rivera); and (8) "'What Can We Learn from Each Other's Experiences?': Observations of a Research-Oriented Workshop by a Classroom Teacher" (Sylvia Robinson). References are included with most papers. (ALF)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED348971

Problem Solving and Critical Thinking for Computer Science Educators

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

- This document has been reproduced as received from the person or organization originating it.
- Minor changes have been made to improve reproduction quality.
- Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

PERMISSION TO REPRODUCE THIS MATERIAL IN **OTHER THAN PAPER COPY** HAS BEEN GRANTED BY
D. Moursund

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

IR015648

NECC monograph

Edited by Cathleen A. Norris and James L. Poirot

ISTE PUBLICATION

2
BEST COPY AVAILABLE

About the Project

Through the efforts of the NECC '90 Conference Chair, Dr. John McGregor, and with financial support provided by IBM and the ACM Precollege Education Subcommittee, two research workshops were added to the NECC '90 agenda. This monograph is a result of one of the research workshops. NECC Research Workshops provide a unique forum for researchers to share ideas in special areas of educational computing. This collection of articles provides an overview of problem solving and critical thinking and its relationship to computing and education.

NECC Monograph Editorial Review Board
Diana Harris, *University of Iowa*
John McGregor, *Clemson University*
Dave Moursund, *University of Oregon*
James L. Poirot, *University of North Texas*

Book Design and Production
Tamara Kidd, *International Society for Technology in Education*
Production Assistance
Ian Byington, *International Society for Technology in Education*

This book has been produced using *Aldus PageMaker*.

© International Society for Technology in Education, 1991
1787 Agate Street, Eugene, Oregon 97403-1923
Order Desk: 800/336-5191
Phone: 503/346-4414 Fax: 503/346-5890
CompuServe: 70014,2117 Internet: ISTE@Oregon.uoregon.edu

ISBN 1-56484-007-7

(IFC) Inside front cover

Problem Solving and Critical Thinking for Computer Science Educators

Edited by Cathleen A. Norris and James L. Poirot

© International Society for Technology in Education, 1991
ISBN 1-56484-007-7

Contents

Problem Solving, Critical Thinking, and Computing: An Overview
Cathleen A. Norris and James L. Poirot 1

***Mindstorms* Revisited: Computers, Problem Solving, and Knowledge-based Instruction**
Karen Swan 9

Defining Programming and Logo as Vehicles for Developing Higher Order Thinking Skills
Jim Dunne 21

Abstracted Knowledge: A Mid-Road Transfer Approach to Critical Thinking
Clifton S. Harris 33

Resolving the Impasse in Software Engineering: Problem Solving in Program Construction
Warren Moseley 39

Critical Thinking and Intelligent Tutoring Systems
James T. Streib 49

Critical Thinking and Open Courseware
Eduardo Rivera 59

"What Can We Learn from Each Other's Experiences?": Observations of a Research-Oriented Workshop by a Classroom Teacher
Sylvia Robinson 65

Preface

Problem solving and critical thinking are the concerns of all teachers at all grade levels and across all curriculum areas. Since 1959, national educational reform has urged the teaching of both of these skills; nevertheless, classroom textbooks and workbook problems often do not require the students to apply problem-solving skills. While classroom teachers are faced with the teaching of problem solving, have these teachers themselves been taught what problem solving actually is? How can teachers use technology to teach problem solving skills in one discipline so that these skills will transfer to another discipline? Do computer programming abilities transfer to problem solving abilities in other disciplines? These are a few of the questions posed at a research-oriented workshop held in June 1990, at the National Educational Computing Conference (NECC) in Nashville, Tennessee.

Research workshops provided a new dimension to NECC activities. Through the efforts of the Conference Chairman, Dr. John McGregor, and with financial support provided by International Business Machines, two research workshops were added to the NECC '90 agenda. The intent was to provide a unique forum for researchers to share ideas in a special area of educational computing. These workshops were not instructional, as are most of the others held in conjunction with NECC. Rather, the leaders of these research workshops served as facilitators. Workshop participants were given the opportunity to present their ideas and findings to a small group of peers with similar research interests.

This monograph is a result of the Problem Solving and Critical Thinking research workshop. Participants responded to a "Call for Participation" by submitting an abstract, and after all abstracts were reviewed, ten participants were invited to attend the one-day workshop which preceded the conference. Four individuals, while not providing abstracts or participating as research presenters, were accepted as observers. These observers, each with unique interests in the workshop topic, provided a valued sounding board for the research participants. Observers included classroom teachers and international visitors.

The eight papers presented in this Monograph were prepared for the workshop leaders after the June meeting and are representative of the participants' breadth of interests. General issues of problem solving and critical thinking in education are included, as well as specialized interests in intelligent tutoring and program construction. Each author includes a valuable bibliography for the interested reader. In addition to the research papers, an article submitted by one of the workshop observers, a classroom teacher, provides an interesting view from a non-research oriented educator.

While this collection of articles does not begin to address all of the issues related to critical thinking and problem solving, it does provide a overview of this pressing issue and its relationship to computing and education. It is hoped that the reader will find it an interesting resource for further work in the area.

Problem Solving, Critical Thinking, and Computing: An Overview

Cathleen A. Norris and James L. Poirot
University of North Texas
Texas Center for Educational Technology

Contributors: Dr. Cathleen Norris is an associate professor in the Department of Computer Education and Cognitive Systems at the University of North Texas and Manager of the Teacher Productivity and Training Laboratory of the Texas Center for Educational Technology. Her research interests include human factors, problem solving, and teacher productivity. Dr. James Poirot is chairman of the Department of Computer Education and Cognitive Systems at UNT and is Manager of the Curriculum and Instruction Laboratory of the Texas Center for Educational Technology. His research interests center around curriculum development, computer-based learning, and teacher education.

Introduction

The teaching of problem solving, critical thinking, and higher order thinking skills is at the top of many educational agendas. Educators have learned that simply having a knowledge of the basics is not sufficient in our ever-changing world. Accumulation of facts and concepts alone does not provide the requisite skills for success. Instead, the ability to solve more and more complex problems is a necessary skill for an "educated" member of today's society. We are, therefore, continually searching for better ways to teach critical thinking, problem solving, and ways to apply specific thinking skills to a variety of situations.

While educators recognize the added importance of teaching problem solving, the body of subject matter content taught in more traditional areas has not decreased. In fact, teachers are finding that not only do they need to teach more history, mathematics, and science, but they must also teach new academic disciplines such as computer literacy and computer programming. It is imperative that a good job be done of directly teaching problem solving and critical thinking skills, while at the same time specific steps must be taken to assure the transference of these skills from one domain to another. Consequently, problem solving and the emphasis upon thinking processes that transfer from one situation to another have become critical components of the curriculum.

Critical thinking, problem solving, reasoning, logical thinking, and higher order thinking skills are terms that are often used interchangeably to refer to a person's ability to analyze a problem situation and come to an appropriate conclusion or solution. This ability is not inherent, and it is estimated that only about twenty-five percent of the first-year college students have the requisite skills for critical thinking (Halpern, 1984). To understand why this is so, it may be helpful to examine some of

the individual areas that researchers have studied in relation to intelligence, higher order thinking, and critical thinking.

Intelligence has been defined as having two main divisions or kinds of activity: crystallized intelligence and fluid-analytic intelligence (Snow, 1982). Crystallized intelligence is that kind which is organized primarily for and by learning in the formal educational medium. Knowledge is organized into formal cognitive systems that can aid learning in the future when a similar educational situation arises. Fluid-analytic intelligence is the reasoning and problem solving ability that begins to develop early in childhood when learning to cope with the natural environment. This skill, which is well developed before schooling begins, continues to develop during the school years, and is characterized by flexible adaptation to novel situations. Although both the crystallized and fluid-analytic capacities can be trained if the educational environment is suitable, each requires a different educational setting, and it is the crystallized type of knowledge that is most often taught.

Smilansky (1984) suggests that intelligence is centered in the ability to solve problems posed by others. He discusses this in the context of a person's ability to invent novel problem solutions and describes a hierarchical relationship between problem solving skill and problem invention ability. He further suggests that while problem solving ability is a necessary component of problem invention, the reverse is not true. A person who can invent problems would have a greater capacity for problem solving than someone who cannot invent a new problem.

Higher order thinking is a term used (Patterson, 1986) to describe the activity of a person engaged in sustained cognitive effort directed at solving a complex problem. Prior knowledge, as well as the effective use of that knowledge, is an important feature of higher order thinking. If the problem is too easy for the person solving it, higher order thinking is not being used; if it is too hard, the person will not be able to solve it. Patterson also discusses the schema theory and its relationship to higher order thinking, illustrating that problem solving skill is dependent in part on a person's ability to draw from a large, problem-specific knowledge base (schemata), and the ability to integrate new information and find new ways of interrelating knowledge and problem solving algorithms. Bransford and Stein (1984) agree with the need for a substantial knowledge of the problem subject area (relevant core concepts) in effective problem solving. He also points out that a person's knowledge of expectations will affect the assessment of the information needed to solve a problem. An awareness of the desired or expected outcome can be used to help determine what information will be needed to help achieve that outcome.

Watson and Glaser (Watson, 1980) view *critical thinking* as a composite of attitudes, knowledge, and skills including: attitudes of inquiry, (i.e., recognizing problems and accepting the general need for evidence in support of what is asserted to be true); knowledge of the nature of valid inferences, abstractions, and generalizations in which the weight or accuracy of different kinds of evidence is logically determined; and skills in employing and applying the above attitudes and knowledge. A major application of critical thinking is problem solving.

Problems and Problem Solving

Everyone has at least a minimum level of problem solving ability. Almost everything one does requires some amount of analysis prior to action. However, there are some people that seem able to solve very complicated problems that stump other generally intelligent people. What techniques and strategies do these people use when achieving their seemingly remarkable solutions, and can others be taught those same techniques? If so, how?

A *problem* may be defined as a "gap" that separates you from a goal (Hayes, 1978), and *problem solving* consists of the process that removes the gap. According to Bransford and Stein (1984), a problem exists whenever the present situation is different from a desired situation or goal. Problems come in two main categories, according to Simon (1973) and Halpern (1984): well structured and ill structured. Well structured problems are those with clearly and thoroughly defined problem states, for which a solution algorithm can be designed. Problems such as calculating satellite orbits or gasoline mileage are well structured. Ill structured problems would be all other types of problems, that is, those that are not well structured. An example might be, "how much does one million dollars weigh?"

Krulik and Rudnick (1980) identify a problem as a situation that requires resolution confronting an individual and for which the individual sees no apparent means or path to obtaining the solution. Here, the phrase "no apparent solution" is important. A question posed to a student, for example, may not be a problem if the answer to the question is obvious to the student.Similarly, exercises given to students in a class may not include problems; exercises often involve drill and practice to reinforce previously learned material. A problem must require thought and synthesis of previously learned knowledge to resolve.

Problem solving is a process of utilizing previously acquired knowledge and skills to address an unfamiliar situation. Problem solving is defined by Nikerson, Parkins, and Smith (1985) as behavior and thought processes directed toward the performance of some intellectually demanding task, while Polya (1945) observes that problem solving happens when a human "attends" to the details of a problem. When attention is focused on the parts of the problem, the problem solving act stems naturally from this attending.

There are two major intellectual activities required for problem solution: understanding and solving. Understanding is the process that helps you comprehend the nature of the gap that separates you from your goal, and the solving process is the execution of procedures that help you bridge the gap. Many identify these procedures in a step-by-step process, with steps taken consecutively. The series of steps often require iteration: that is, one must recycle through the problem solving process to improve upon the arrived at solution.

Problem solving, as defined by Bransford and Stein (1984), consists of five steps:

i. identify the problem
ii. define and represent the problem
iii. explore possible strategies

iv. act on the strategies
v. look back and evaluate the effects of your actions.

Halpern (1984) uses similar divisions with a unique, and not always used, fourth step:

i. preparation or familiarization - understanding the problem space, goals, etc.
ii. production—produce the various solutions
iii. judgement or evaluation—decide which of the possible solution paths to follow
iv. incubation—a "dormant" period following concentrated effort, when solutions sometimes present themselves without prompting.

Polya (1945) defines problem solving as having four steps:

i. understand the problem
ii. devise a plan
iii. carry out the plan
iv. look back at the results and revise as needed.

While considerable work has been accomplished in identifying the steps necessary to solve a problem, we are also interested in determining if problem solving skills learned in one discipline transfer to skills in another. For example, does the study of problem solving in mathematics improve upon one's ability to solve problems in other areas such as financial planning or real estate?

Transfer of Problem Solving Ability

There is a substantial interest in teaching a general level of critical thinking that would allow a person to solve problems in a wide variety of situations. The idea that problem solving skills learned in one context will be usable in another is known as transference or generalization and has been studied many times, mostly from the viewpoint of trying to teach critical thinking directly. Transference is not easily achieved and is especially difficult when processes are not appropriate. Transference of critical thinking frequently fails because the underlying processes are seldom made explicit (Linn, 1985). Traditionally, students have been encouraged to "think", but have not been taught how to accomplish conscious thinking.

Transference usually occurs best between tasks that are very similar in nature (Snow, 1982; Kuland, 1986; Linn, 1985). Most researchers have found that the skills that transfer best are those that are the least problem or domain specific. Patterson (1986) found that training in multiple contexts helped decrease a student's tendency to use an algorithm or heuristic only in the context in which it was learned.

Nikerson, Parkins, and Smith (1985) point out that a traditional scenario would have the student learn a skill in one context or educational setting, and then be tested for transferrence in another context. Typically, there is a wide gap between the two contexts and generalization is inhibited.

The ability level of the problem solver may also affect generalization. Novice problem solvers approach problem solving very differently from expert problem

solvers. Novices tend to focus more narrowly on syntax, form, and other surface features. Experts focus more on the larger problem classification, functional similarities with previously solved problems, and other deep structure features of the problem. These different problem viewpoints will cause novices and experts to transfer their skill to other domains differently.

Related to the differing viewpoints of novices and experts is the issue of how each person organizes knowledge. Adelson (1981) discusses a hierarchical arrangement of chunks, with each chunk representing some relevant type of problem description criterion. In the case of programming, these chunks might be loops, procedures, subprocedures, or any other unit that has some functionally unique quality. Experts group these chunks in a hierarchy so that later recall will find the highest level relevant chunk. Novices, on the other hand, do not develop hierarchies at first, or the hierarchies are very simple. Consequently, a novice either must take much longer to recall relevant information, or must synthesize a new problem solution for each new situation because s/he doesn't recognize any matching pattern. This, of course, would detrimentally affect any hoped-for transfer of problem solving strategies.

While transferance is an important issue no matter what curricular area is being discussed, our major concern is computer education and its impact upon problem solving abilities. This topic has been the object of recent research efforts.

Computing and Problem Solving

Why do we teach topics in computing? There are obviously a variety of answers to this question including "to prepare for a career in computing" and "to prepare for a career where computing skills, such as word processing, are a necessity." We also often hear that the study of computing, in particular computer programming, improves upon one's ability to solve problems. Is this, in fact, a consequence of computing instruction?

Mayer, Dyck, and Vilberg (1986) examined transference in computer programming and studied improvements in problem solving in domains beyond the programming language that is taught. They indicate that a recent curriculum designed specifically to teach general thinking skills has not produced any transfer to other domains, and further, that students did not show any statistically significant improvement in general thinking ability after taking a BASIC programming course.

Clements (1984) found that there were no differences between the general cognitive developments of children who were taught LOGO programming versus those of children who used CAI programs that taught problem solving, but there was improvement by the LOGO group in some specific thinking skills areas such as metacognition, creative thinking, and divergent thinking.

Kurland, Pea, Clement, and Mawby (1986) say that it is primarily sophisticated programming activities that require higher level thinking. This implies that introductory programming problems may not be sufficiently difficult to elicit substantial critical thinking. "Sophisticated," however, may be relative to the current level of expertise of a given programmer, as discussed by Patterson and Smith (1986).

One problem unique to computer utilization and critical thinking transference at

the beginner level is that students must acquire familiarity with several formal systems before they can begin associating similarities among systems and applying analogous problem solving skills across those systems (Linn, 1985). Thus, a student beginning the study of computing must advance in knowledge and experience on computers before any demonstrable, wide ranging problem solving skills can be noted. It is similarly claimed that problem solving expertise accrues only after many hours of practice in a given domain (Frederiksen, 1984). Therefore, this amount of practice, and the acquisition of a broad range of formal system familiarity, might not be possible in a single semester programming class.

Palumbo (1990) extensively reviewed literature that related to problem solving and computer programming. While the computer has often been touted as a tool that will greatly assist in developing good problem solving skills, Palumbo found little evidence supporting this claim. Concerns about research consistency and relevance center around variation in methods of teaching programming, inadequate length of treatment (instructional time), lack of intensity of treatment (such as hands-on computer time, rigor of instruction, and proximity of instructional sessions to each other), and choice of programming language used for instruction. Such concerns point to a need to continue with well-designed research efforts.

What's Next?

Problem solving is a necessary precursor to success in most professions, as well as in everyday living. While it seems that we have a good handle on what a problem is and what processes are necessary for solving problems, we do not have adequate information on the transference of problem solving from the study of computing to other problem situations. Researchers in the field might wish to continue addressing questions such as:

- Can we teach problem solving directly?
- Can we prepare teachers to teach problem solving better?
- How do we measure success of teaching problem solving?
- Are there particular topics in computing which lend themselves to improvement of problem solving and critical thinking?
- Will the teaching of computer programming improve one's general problem solving ability?
- When teaching topics in computing, what techniques or educational practices will enhance transference of problem solving?
- What makes for a good problem solver?

It is obvious that our society is becoming more and more computer literate. It is less obvious that teaching certain computing topics improves one's ability to function in today's society. Continued use of computers in education might very well center upon our ability to demonstrate improvement in problem solving skills through computer instruction.

References

Adelson, B. (1981). Problem solving and development of abstract categories in programming languages. *Memory and Cognition, 9*(4).

Bransford, J., & Stein, B. (1984). *The ideal problem solver.* New York: W.H. Freeman and Company.

Clements, D., & Gullo, D. (1984). Effects of computer programming on young children's cognition. *Journal of Educational Psychology, 76,* 1051-1058 .

Cronbach, L. (1970). *Essentials of psychological testing* (3rd ed.). New York: Harper and Row.

Frederiksen, N. (1984). Implications of cognitive theory for instruction in problem solving. *Review of Educational Research, 54*(3), 363-407.

Halpern, D. (1984). *Thought and knowledge: An introduction to critical thinking.* Hillsdale, NJ: Lawrence Erlbaum and Associates

Hayes, J. (1978). *Cognitive psychology: Thinking and creating.* Homewood, IL: Dorsey Press.

Krulik, S. & Rudnick J. (1980). *Problem solving: A handbook for teachers.* Boston, MA: Allyn and Bacon.

Kurland, D. M., Pea, R. D., Clement, C., & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research, 2*(4), 429-458.

Linn, M. (1985). The cognitive consequences of programming instruction in classrooms. *Educational Researcher, 14*(5), 14-29.

Mayer, R., Dyck, J., & Vilberg, W. (1986). Learning to program and learning to think: What's the connection? *Communications of the Association for Computing Machinery, 29*(7), 605-610.

Nikerson, R., Parkins, D., & Smith, E. (1985). *The teaching of thinking.* Hilsdale, NJ: Lawrence Erlbaum Associates.

Patterson, J., & Smith, M. (1986). The role of computers in higher order thinking. In J. Culbertson, & L. Cunningham (Eds.), *Microcomputers in education: 85th Yearbook of the National Society for the Study of Education.* Chicago, IL: University of Chicago Press.

Palumbo, D. B. (1990). Programming language/problem-solving research: A review of relevant issues. *Review of Educational Research, 60*(1), 65-89.

Polya, G. (1945). *How to solve it.* Princeton, NJ: Princeton University Press.

Smilansky, J. (1984). Problem solving and the quality of invention: An empirical investigation. *Journal of Educational Psychology, 76,* 377-386.

Snow, R.E. (1982). The training of intellectual aptitude. In D. K. Detterman, & R. J. Sternberg (Eds.), *How and how much can intelligence be increased*? Ablex Corporation.

Simon, H. (1973). The structure of ill structured problems. *Artificial Intelligence, 4.*

Watson, G., & Glaser, E. (1980). *Watson-Glaser critical thinking appraisal manual.* San Antonio, TX: The Psychological Corporation.

Mindstorms Revisited: Computers, Problem Solving, and Knowledge-based Instruction

Karen Swan
State University of New York at Albany

Contributor: Karen Swan is a professor at State University of New York at Albany. Her research interests include cognition and computing, hypermedia and visual literacy.

Introduction

In 1980, Seymour Papert published *Mindstorms*, a book that excited many educators with its notion that Logo programming was fertile ground for the development of generalized problem solving and critical thinking abilities. In this regard, Papert made two claims. First, he argued that the computer was a revolutionary educational tool because it supported "transitional objects to think with" (Papert, 1980, page 21), quasi-concrete representations of abstract procedural ideas that could help learners "bridge the gap between concrete and formal thought." Second, he argued that the transition from concrete to formal thinking would take place "painlessly, successfully, and without organized instruction" (Papert, 1980, page 9) when learners were given computing environments rich in such transitional objects to explore, environments like the one he helped develop specifically for that purpose: the Logo programming language.

. These two claims have engendered enormous amounts of research and heated debate both within and outside of the educational computing community. The research reviewed in this paper was designed to investigate each of the claims individually in relation to the teaching and learning of particular problem solving strategies through Logo programming experience. The utility of separating the claims is born out by our results, which indicate that the first claim—that computing environments can uniquely support the development of problem solving skills—is true; whereas the second claim—that this development will happen automatically—is not. These results make sense of the seemingly contradictory reports found in the literature concerning Logo programming and the teaching and learning of problem solving. It can be shown that investigations that looked for the automatic development of problem solving skills resulting from Logo programming experience (Papert, Watt, diSessa & Weir, 1979; Pea & Kurland, 1984; Clement, Kurland, Mawby & Pea, 1986; Leron, 1985; Shaw, 1986; Mandinach & Linn, 1987) found no such occurrence, but that those that investigated mindful interventions combining explicit instruction with Logo programming practice (Clements & Gullo, 1984; Clements, 1987; Delclos, Littlefield & Bransford, 1985; Carver & Klahr, 1986; Carver, 1987; Lehrer & Randle, 1987; Lehrer, Sancilio & Randle, 1989; Thompson & Wang, 1988; Swan & Black, 1988; Swan, 1989; Swan, 1990; DeCorte, Verschaffel & Schrooten, 1989) reported positive findings.

Our research results do more than settle a long-standing argument among computer educators. They do more than provide a blueprint for teaching problem solving through Logo programming. By separating Papert's claims and experimentally examining each in itself, we have begun a process that can help us understand how computers can uniquely support the development of higher order cognitive skills, a process that will, we hope, shed some light on the mechanisms involved in that development as well. Indeed, our results suggest that there are two kinds of knowledge involved in the application of problem solving and critical thinking skills, both of which must be addressed in their teaching and learning. These are procedural knowledge, or what Gilbert Ryle (1949) called "knowing how," and declarative knowledge, or what Ryle called "knowing what."

"Knowing How": Computer-Supported Objects to Think With

A quarter of a century ago, Marshall McLuhan (1964) advanced the notion that the communications media were extensions of the human sensory apparatus which altered the ways in which we use our senses, and so, the manner in which we perceived ourselves and our environment. McLuhan argued that the formal nature of a particular medium of communication was, therefore, more influential in shaping our thought than the content of the message it transmitted. The development of language, for example, altered the way we perceive the world in that the naming and defining of objects in it allows us to distinguish between those objects in ways that would not be possible without language. The naming and defining aspects of spoken language were, in turn, internalized as cognitive functions, making possible such mental operations as sorting and classifying, and changing the nature and structure of the way we think.

Whether or not one accepts McLuhan's hypothesis in toto, it seems clear that each medium of communication entails unique formal attributes that matter, or that can be made to matter, in learning. Gavriel Salomon (1981), for example, has shown how differing filmic presentations can "activate," "short-circuit," or "model" particular cognitive processes. In a similar vein, Seymour Papert has argued that the formal properties of computers are particularly supportive of the development of problem solving abilities.

Papert maintains that computers are unique educational tools because they support transitional objects to think with. His idea seems to be that because computing environments support the building of interactive procedural models, they allow the representation, manipulation, and dynamic testing of abstract procedural ideas in ways that are not possible in the more static media. As well as relieving burdens to working memory that might otherwise overwhelm students, the use of such models eases their internalization. Computer-based models of complex interactive procedures thus can provide a scaffolding for the internalization of procedural knowledge, of "knowing how" (Salomon, 1988), and so help bridge the gap between concrete and formal thought (Papert, 1980).

Subgoals formation, for example, is a problem solving strategy that is given quasi-concrete representation in Logo programming because programs are composed

of small subprocedures, each of which defines the means to satisfying particular parts of a larger programming problem. Each subprocedure, moreover, can be written, tested, and refined by itself before the parts are assembled to create the larger program. Programming in this manner, it can be argued, provides a quasi-concrete model of the abstract process of subgoals formation with which students can interact. This sort of interaction leads to the internalization of subgoals formation processes in much the same way that the sorting and classifying of objects in the real world leads to the internalization of sorting and classifying procedures.

Our research suggests that Logo programming can model at least four particular problem solving strategies in this manner—subgoals formation, forward chaining, systematic trial and error, and analogy. In three separate studies involving students ranging in age and ability from the fourth through twelfth grades, we found that students practicing such strategies through Logo programming improved on paper and pencil tests of their ability to apply the strategies, whereas students working in other media did not (Swan & Black, 1988; Swan, 1989; Swan, 1990).

Figure 1 shows a comparison of the pre- to post-test changes in the scores of two groups of fourth to sixth grade students on four problem solving strategy tests. Both groups were given the same explicit problem solving instruction. The students in one group worked on Logo programming practice problems, while students in the other group were given similar practice problems involving cut-paper manipulatives. Significant differences in learning were found between groups ($F_{2,97} = 12.96$; $p < .01$). Those differences favored the Logo programming practice group, indicating that students in that group internalized the problem solving strategies, whereas students working with a more concrete medium did not.

Moreover, it can be argued that students showed the greatest improvements on precisely those strategies (subgoals formation and systematic trial and error) that can be most concretely modeled in the Logo programming environment. Indeed, the one problem solving strategy we investigated for which no increases were found was the one strategy that has no Logo programming analog at all—backward chaining. There is no way a student can write a Logo program backwards, thus there is no concrete Logo programming model of the backward chaining process. This relationship between the degree to which problem solving strategies can be modeled in Logo programming and the degree to which students in our studies improved in their general ability to apply those strategies suggests that students' increased problem solving abilities resulted from their internalization of those models.

The results of our research thus support Papert's claim that computing environments have unique formal properties that can support the development of problem solving and critical thinking skills, hence, that they might play, as he has argued, an important role in the teaching and learning of such skills. In particular, our research suggests that computing environments can uniquely model higher order cognitive processes in ways that ease their internalization. Those ways seem to involve computers' peculiar interactivity.

It stands to reason that an interactive medium is better suited for the representation

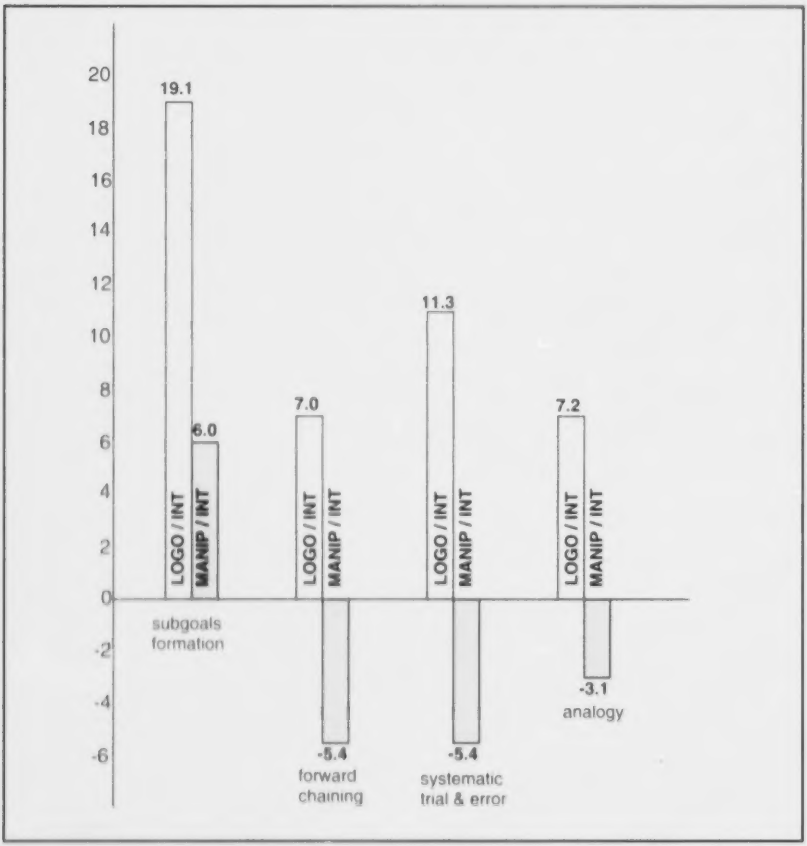

Figure 1. Comparison of Mean Pre- to Post-test Differences: Logo Programming vs. Concrete Manipulatives Practice Environment

of complex interactive processes like problem solving than are the traditional static media. We have seen how the specific steps of particular problem solving strategies can be instantiated as corresponding parts of Logo problem solutions, and that such programming experience can result in increased general problem solving abilities (Swan & Black, 1988; Swan, 1989; Swan, 1990). Other researchers have shown how Logo-based geometry (Lehrer, Sancilio & Randle, 1989) and Cartesian coordinate (Thompson & Wang, 1988) microworlds can be created that model and so develop students' conceptual thinking in those domains. Salomon (1988) demonstrated that the use of a computer-based reading aid that explicitly models reading-related metacognitive strategies can result in improved reading comprehension among low ability students. In all these cases, students' higher order cognitive skills were

enhanced through their experience with interactive computer-based programs that explicitly modeled those skills, whereas students involved with similar static models showed no such skill increases.

"Knowing What": Knowledge-Based Instruction

It is important to realize, however, that in all the aforementioned cases, the cognitive skills in question were explicitly named and defined, either by the programs in combination with explicit instruction and/or instructional materials, or by explicit instruction embedded in the programs themselves. In our studies, we found that students did not automatically improve on problem solving strategy measures as a result of Logo programming experience, but rather that they did so only when those strategies were explicitly taught (Swan & Black, 1988; Swan, 1989; Swan, 1990).

We found, for example, that fourth through sixth grade students given Logo programming problems to solve in a discovery learning condition did not improve on problem solving strategy measures, whereas students given explicit problem solving instruction and similar problems to solve did ($F_{2,97} = 12.96$; $P < .01$). Figure 2 shows the differences between the pre- to post-test changes for both groups. Likewise, high school students given explicit problem solving instruction with Logo programming practice improved on the strategy measures, whereas students given traditional instruction in subjects commonly recommended for the development of problem solving skills (pre-Calculus and Advanced Placement Pascal programming) did not ($F_{2,37} = 11.56$; $p < .01$). Figure 3 details those differences. Sharon Carver has reported similar findings regarding the teaching and learning of debugging strategies through Logo programming experience (Carver & Klahr, 1986; Carver, 1987).

Such findings are really nothing new; they mirror Thorndike's (1907) concerning Latin and logical thinking. Thorndike concluded that higher order skills would only be internalized and transferred to other domains to the extent that such skills were explicitly taught. Salomon and Perkins (1987) argued that higher order skills are by definition not skills that can be automated, but rather are skills that must be mindfully applied. For this reason, they, too, maintain that higher order skills must be explicitly taught if they are to be generalized and applied in differing domains. To explicitly teach higher order skills, one needs to specify the knowledge those processes entail; one needs to premise the design of instruction on knowledge outcomes, rather than on behavioral outcomes. The distinction, we believe, is a real one.

Problem solving instruction, for example, has traditionally sought behavioral outcomes. It has focused on the increased ability to solve particular kinds of problems and has accordingly concentrated on practice solving those problems. While practice alone may be enough for what Salomon and Perkins (1987) call "low road" learning, for automating lower-level cognitive skills, it is insufficient for the learning of processes that must be mindfully applied. Indeed, as Thorndike demonstrated, this doesn't work. While students in his studies might have become better at solving particular problems in Latin, they did not acquire generalized logical thinking skills that could be transfered to problems in differing domains. Similarly, students in the

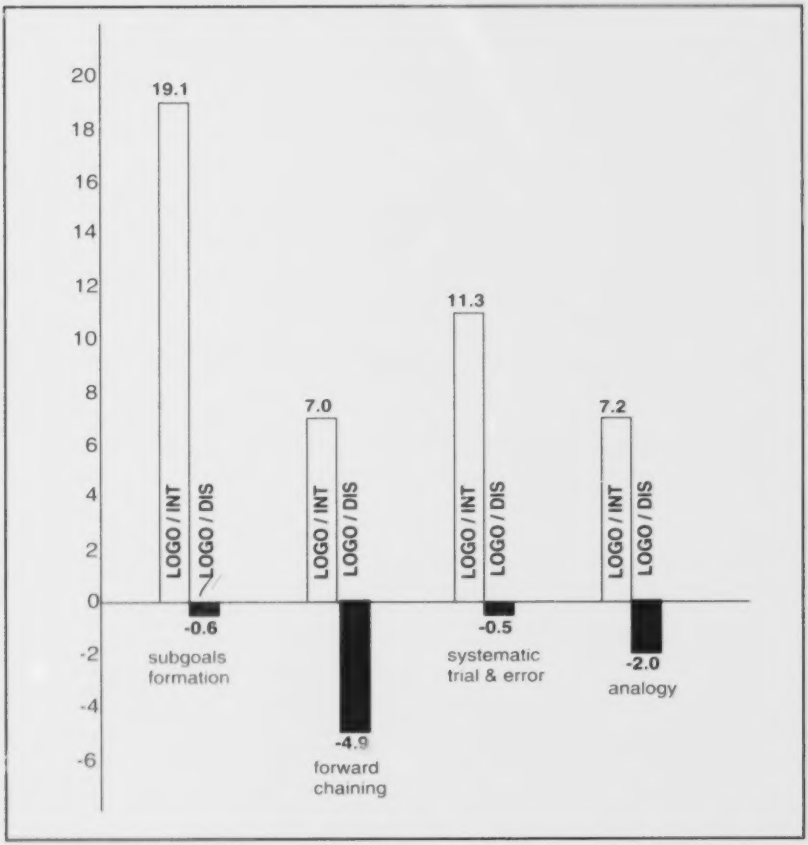

Figure 2. Comparison of Mean Pre- to Post-test Differences: Explicit Training vs. Discovery Learning

early Logo studies might have become better at solving particular Logo programming problems, but those skills were not generalized and transfered to other domains.

We believe what is at issue here is the abstraction and generalization of knowledge. We believe that it is not enough to give students procedural experience solving particular problems, not even when that experience includes working with powerful, interactive models of problem solving processes, because the experience is perceived as domain specific and is, therefore, neither abstracted nor generalized. The naming and defining of the steps involved in particular problem solving strategies, however, are not domain specific. Just as the naming and defining of objects through language lead to the abstraction of the qualities of those objects so that they can be manipulated as objects of thought, so the naming and defining of particular problem

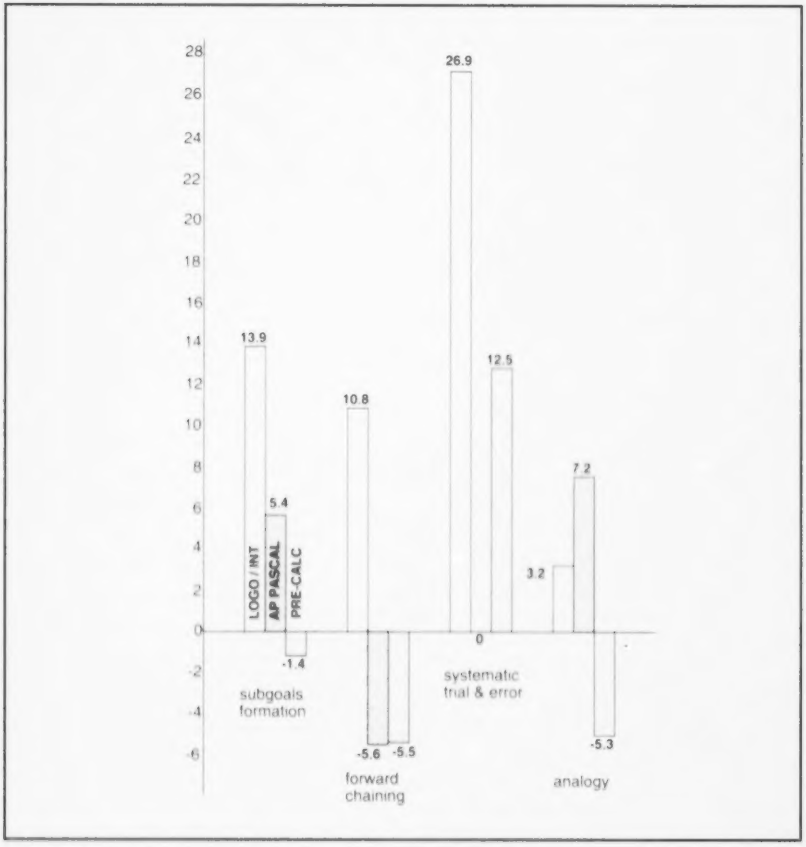

Figure 3. Comparison of Mean Pre- to Post-test Differences: Explicit Training with Logo Programming Practice vs. Traditional Subject Area Instruction

solving strategies decontextualize those strategies so that they can be generalized and applied to problems in other domains.

We call this approach "knowledge-based instruction." This is what we did in designing the Logo-based problem solving intervention we used in our studies. We performed task analyses of the six particular problem solving strategies we believed would be most applicable to Logo programming: subgoals formation, forward chaining, backward chaining, systematic trial and error, alternative representation, and analogy; and we used the task analyses to name and define the strategies for our students. For each strategy we made wall posters (Figure 4) that enumerated the steps involved in them and used these to introduce each strategy unit. Only after providing students with this declarative knowledge base did we have them apply it to Logo

SUBGOALS FORMATION

1 What is the problem?

2 What little problems are a part of the problem? Make a tree.

3 Can you solve them? If yes, solve them and use your tree to reassemble the parts.

4 If no, go back to 2 and redo 2 and 3 for each of your smaller problems.

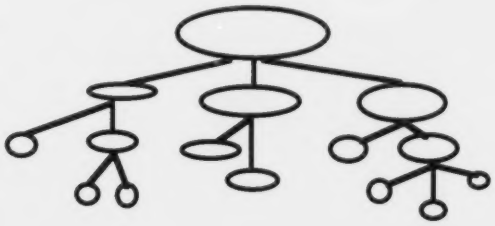

Figure 4. Wall Chart Showing Steps in Subgoals Formation

programming problems particularly amenable to solutions involving the use of the strategy under discussion. We believe that the declarative knowledge of particular problem solving strategies provided the students in our experimental groups with "pointers" to the procedural models of those strategies acquired through Logo programming. Those pointers allowed the students to retrieve and apply the strategies to solve the paper and pencil problems on which they were subsequently tested.

Indeed, our most recent study focused specifically on the role of the declarative knowledge of particular problem solving procedures in the development of corresponding generalized procedural skills. In it we randomly assigned fourth and fifth grade students to one of two groups. Both groups received exactly the same Logo programming instruction and exactly the same programming problems to solve. The

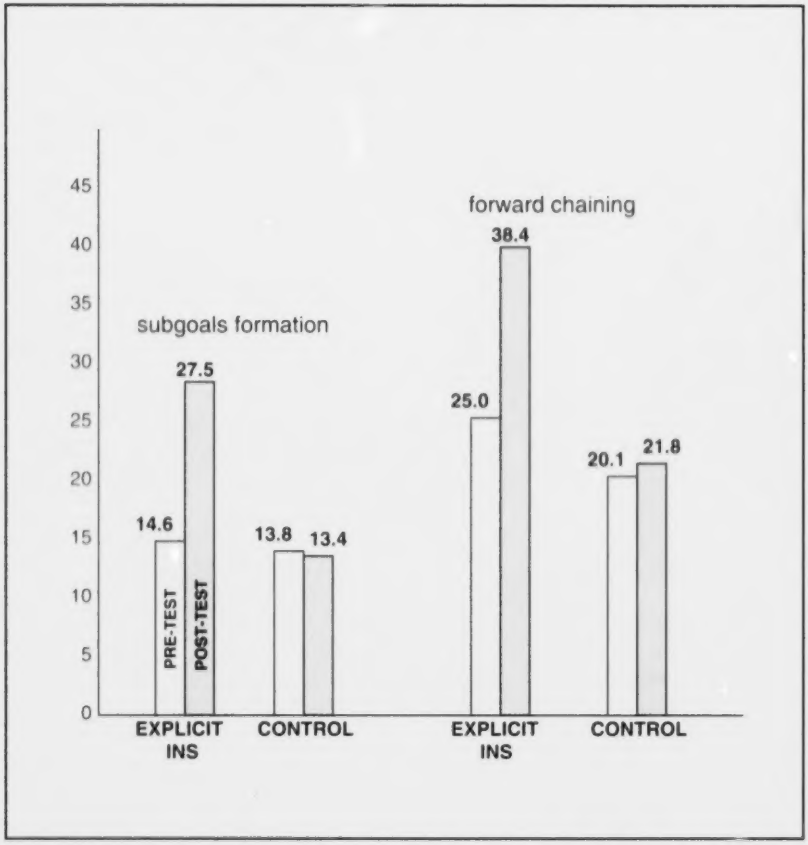

Figure 5. Comparison of Mean Pre- and Post-test Scores Explicit Problem Solving Instruction vs. No Explicit Instruction

only difference between the two groups was that students in one group received explicit instruction on the steps involved in the two problem solving strategies we tested—subgoals formation and forward chaining—whereas students in the other group received no such instruction. Figure 5 compares mean pre-and post-test scores on the two strategy measures for students in both groups. It shows that both the subgoals formation and forward chaining skills of the students receiving explicit problem solving instruction improved, while those of the students who did not receive explicit instruction did not. The results indicate that "knowing how" is not enough to insure the generalization and transfer of complex higher order cognitive processes; that for such knowledge to be abstracted from a particular domain and mindfully applied in another, one must also know precisely "what" is being abstracted and applied.

Mindstorms Revisited

The research reviewed in this paper demonstrates the effectiveness of a specific instructional approach for developing particular problem solving strategies. More importantly, it indicates that the Logo programming environment in particular and computing environments in general can, as Seymour Papert (1980) argued, uniquely support the development of higher order cognitive processes. The reason computing environments can be uniquely supportive of such development seems to be that they can model procedural knowledge ("knowing how") in ways that ease its internalization. Those ways appear intimately tied to computers' peculiar interactivity.

On the other hand, the research contradicts Papert's claim that the development of higher order cognitive skills will happen automatically when students explore environments rich in such cognitive models. Rather, it indicates that students must be explicitly instructed in the specific steps those models entail before they can abstract and mindfully apply them in differing contexts. That explicit instruction seems necessary suggests a declarative ("knowing what") component to complex procedural knowledge. It is possible that such declarative knowledge provides "pointers" that help us to access and mindfully apply internalized procedures.

The development of higher order cognitive skills has been given top priority in American education. If computing environments can indeed be designed to support such development, they might play an important role in its teaching and learning. Several notions have been developed in this paper that might guide educators seeking to use computers for these purposes. In particular, we suggest that any such intervention should be knowledge-based; that it should combine instruction aimed at developing students' declarative knowledge of higher order cognitive processes with practice aimed at developing students' internal procedural models of those processes. Also, that practice should include experience with interactive computer programs that concretely model the processes in question. In today's educational climate these notions certainly deserve further investigation.

References

Carver, S. M. (1987). Transfer of Logo debugging skill: Analysis, instruction, and assessment. *Computer Systems Group Bulletin, 14*(1), 4-6.

Carver, S. M., & Klahr, D. (1986). Assessing children's Logo debugging skills with a formal model. *Journal of Educational Computing Research, 2*(4), 487-525.

Clement, C. A., Kurland, D. M., Mawby, R., & Pea, R. D. (1986). Analogical reasoning and computer programming. *Journal of Educational Computing Research, 2*(4), 423-454.

Clements, D. H. (1987). Longitudinal study of the effects of Logo programming on cognitive abilities and achievement. *Journal of Educational Computing Research, 3*(1), 73-94.

Clements, D. H., & Gullo, D. F. (1984). Effects of computer programming on young children's cognition. *Journal of Educational Psychology, 76*, 1051-1058.

DeCorte, E., Verschaffel, L., & Schrooten, H. (1989). *Cognitive effects of learning to*

program in Logo: A one-year study with sixth graders. Leuven, Belgium: Project: Computers and Thinking, Center for Instructional Psychology, Katholieke Universiteit Leuven.

Delclos, V. R., Littlefield, J., & Bransford, J. D. (1985). Teaching thinking through Logo: The importance of method. *Roeper Review, 7*(3), 153-156.

Lehrer, R., & Randle, L. (1987). Problem solving, metacognition and composition: The effects of interactive software for first-grade children. *Journal of Educational Computing Research, 3*(4), 409-428.

Lehrer, R., Sancilio, L., & Randle, L. (1989). Learning pre-proof geometry with Logo. *Cognition and Instruction, 6*(2), 159-184.

Leron, U. (1985). Logo today: Vision and reality. *The Computing Teacher, 12*(6), 26-32.

Mandinach, E. B., & Linn, M. C. (1987). Cognitive consequences of programming: Achievements of experienced and talented programmers. *Journal of Educational Computing Research, 3*(1), 53-72.

McLuhan, M. (1964). *Understanding media*. New York: New American Library.

Papert, S. (1980). *Mindstorms*. New York: Basic Books.

Papert, S., Watt, D., diSessa, A., & Weir, S. (1979). *Final report of the Brookline Logo Project* (Logo Memo 53). Cambridge, MA: MIT Artificial Intelligence Laboratory.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2*(2), 137-167.

Ryle, G. (1949). *The concept of mind*. New York: Hutchinson.

Salomon, G. (1981). *Interaction of media, cognition, and learning*. San Francisco: Jossey-Bass.

Salomon, G. (1988, April). *AI in reverse: Computer tools that become cognitive*. Paper presented at the Annual Meeting of the American Educational Research Association, New Orleans.

Salomon, G., & Perkins, D. N. (1987). Transfer of cognitive skills from programming: When and how? *Journal of Educational Computing Research, 3*(2), 149-170.

Shaw, D. G. (1986). Effects of learning to program a computer in BASIC or Logo on problem solving abilities. *AEDS Journal, 19*(2/3), 176-189.

Swan, K. (1990). Programming objects to think with: Logo and the teaching and learning of problem solving. *Journal of Educational Computing Research, 6*(4).

Swan, K. (1989). Logo programming and the teaching and learning of problems solving. *Journal of Artificial Intelligence in Education, 1*(1).

Swan, K., & Black, J. B. (1989). *Logo programming and the teaching and learning of problem solving* (CCTE Report 89-1). New York: Teachers College, Columbia University.

Thompson, A. D., & Wang, H. M. C. (1988). Effects of a Logo microworld on student ability to transfer a concept. *Journal of Educational Computing Research, 4*(3), 335-347.

Thorndike, E. L. (1907). *Elements of psychology*. New York: Teachers College Press.

Defining Programming and Logo as Vehicles for Developing Higher Order Thinking Skills

Jim Dunne
Long Island University

Jim Dunne is a professor in the Department of Educational Technology at Long Island University. His research interests include learning environments and technology and teacher education.

Introduction

For the past decade I have been actively involved in promoting the use of the computer programming language Logo as a vehicle for helping children develop higher order thinking skills. During this time, I've experienced the initial excitement and optimism generated by Logo's potential for changing and improving the way teachers teach and children learn. I've seen Logo's rapid and widespread acceptance and adoption by schools throughout the country. I've noted the gradual disillusionment among many early advocates of Logo as they began to believe that it wasn't delivering on its promise. And currently, I'm seeing how Logo is being adapted to blend in with more traditional modes of instruction.

This scenario is not new. Educational radio, film, and television were all introduced with extraordinary claims for how they would change teacher practice and enhance student learning (Cuban, 1986, page 4). Yet these potentially *revolutionary* technologies were not flexible enough to be easily adapted to meet traditional curricular or teacher needs. Consequently, even when adopted, they were rarely used. This cycle of hope to disillusionment occurs throughout the literature of educational change and the history of instructional technology. It starts with high hopes for easy and radical change fueled by the oversimplification and the oversell of the new reform or technology. This is followed by the reality and problems of implementation and the resistance to changing familiar modes of instruction and practice. Classroom experience is usually corroborated by theoretical and empirical criticism that *it* didn't *deliver*, or that the *system* doesn't have the capacity for real change.

The most successful educational innovations demand the least amount of change in the organizational structure of a school and in the behavior of teachers and students. The most common current use of computers in classrooms is for drill and practice. The content and methodology used in this technology is familiar to all teachers. It does not require them to obtain new knowledge or skills, and it can be easily integrated into mainstream classrooms. Logo, on the other hand, usually requires organizational, curricular, methodological, and behavioral changes in the classroom. The complexity of creating a *Logo culture* will always be a serious obstacle to successful implementations. But my own experience and recent literature indicate that when these changes are made, Logo can deliver on its promise.

The difficulty of implementing Logo needs to be weighed in view of its potential for helping children develop better problem solving and thinking skills. The same decade in which Logo has been available to schools has also been a time of major concern about the effectiveness of American education. Dozens of reports and national commissions have documented a system of schooling that is failing to meet the needs of a modern society. Each report has a unique purpose and focus. They don't always agree on the causes and severity of the problem, and on what needs to be done to correct it. Yet there is a consensus that schools need to create and emphasize better ways to help students develop the problem solving, critical, and creative thinking skills necessary to function successfully in a technology- and information-based economy.

Logo is not the only or the best way to help children develop these skills. But it is still in widespread use, usually with the expectation that it will lead to the development of higher order thinking. After ten years of teaching teachers about Logo, I've realized that if it is to have any lasting and beneficial effect, those charged with implementing Logo need to have a clear, balanced, and realistic sense of what Logo is and what is required to implement it.

This paper is an attempt to define Logo as a vehicle for the development of higher order thinking skills. Logo is defined both as a programming language and as a learning culture. Its roots are directly traced to the work of Jean Piaget and to the computer science subfield of artificial intelligence. Logo is described as the basis of a child-centered exploratory learning environment where the child's explorations are intended to foster the development of problem solving skills, higher order thinking skills, mathematical skills, and the type of self-reflective behaviors that will lead the child to a better understanding of, and control over, his or her own thought processes.

What is Logo?

Logo is a computer programming language.
Logo is a high-level computer language.
Logo is a graphics language.
Logo is a discovery-based language.
Logo is turtle graphics.
Logo is turtle geometry.
Logo is a list-processing language.
Logo is a collection of objects to think with.
Logo is a mathematics microworld.
Logo is a computer language that promotes problem solving.
Logo is a computationally rich exploratory environment.
Logo is a powerful programming language.
Logo is a high-level computer language designed for children.
Logo is obsolete.
Logo is many things to many people.

The above responses by elementary and secondary school teachers to the question, "What is Logo?" (given at the start of introductory and advanced level Logo

graduate courses) indicate the difficulty in precisely defining Logo and, in turn, foreshadow many of the difficulties associated with its implementation in schools.

There is little disagreement on the definition of Logo as a programming language: it is interpreted (translated to machine language during execution), procedural (complex operations are created by combining simpler ones), and extensible (commands can be defined in a way that will make them equivalent to the primitives of the language). Harvey (1985) describes Logo as a very powerful language, at least as far as languages available for personal computers go, in that it is designed to let the user concentrate on the actual problem to be solved, rather than spending a disproportionate amount of time worrying about the constraints (syntax and semantics) of the language. One of the guiding principles used in the development of Logo was that it should have no threshold and no ceiling: no threshold in the sense that users should not need command of much of the syntax and semantics (the commands and how they're organized) of the language before they can do personally meaningful things with Logo, and no ceiling in the sense that as users' expertise and sophistication in programming grows, the language will not impose unnecessarily frustrating limits on their ability to accomplish their goals.

Given the current state of the art of computers and the people who program and use them, the goal of no threshold/no ceiling is still something to strive for. Logo and computer applications with similar philosophies are a first step at providing computer environments where the major emphasis can be focused on the task, not on the tool used to accomplish the task.

Logo's Immediate Roots

The ideas behind Logo were developed in the mid 1960s at Bol Beranek and Newman in Cambridge, MA, but soon found a permanent home in the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology under the direction of Prof. Seymour Papert. The initial goal was to put powerful computing (of the kind that was at the time available only to "real" computer scientists) into the hands of children and novices. Papert (1980a) states that Logo's roots are in the work of Jean Piaget and from the computer science subfield of artificial intelligence (AI).

Piaget

From the work of Piaget that centers on the development of intelligence, Logo stresses the importance of developing a rich environment, or culture, in which children can play, explore, and learn in a natural, spontaneous way. While the work of Piaget is subject to a variety of interpretations, it stresses the qualitative changes or stages of thought that a learner progresses through as he or she is presented with different intellectual challenges (Piaget, 1972; Favell, 1963). Each stage can be identified by the strategy learners use to interact with and represent phenomena they encounter in the real world. The focus is the cognitive experience that allows a learner to "assimilate" new knowledge to form theories and explanations of how the world works. Then they "accommodate" this knowledge when they modify or abandon

particular explanations or theories based on new experiences and more powerful explanations. While many interpretations of this theory prescribe a rigid chronological span to each stage, Papert and several current Piagetian researchers (e.g., Case, 1978; Feuerstein, 1980) stress the process of development rather than the stages themselves. Their emphasis is on the quality and quantity of the experiences learners have with particular strategies, and in optimizing the conditions under which they encounter this new knowledge and apply their strategies.

Artificial Intelligence

The field of artificial intelligence lacks a precise definition. Winston (1977) defines AI as "the study of ideas which enable computers to do things that make people seem intelligent" (page 1). Rich (1982) defines AI as "the study of how to make computers do things at which, at the moment, people are better" (page 1). Papert (1980a) says that AI, narrowly defined, "is concerned with extending the capacity of machines to perform functions that would be considered intelligent if performed by people" (page 157). In order to create smart machines, AI researchers need to understand and reflect on the "intelligent" things they want the computer to do. This reflection involves the attempt to give concrete form to ideas about thinking and intelligence, and a notion that you learn most from instances when your expectations fail. Schank and Slade (1985) believe that this concept of expectation failure can provide important opportunities for learning. They state that when an expectation fails, you are stimulated to explain what went wrong, and that these explanations can be used to locate previous, related experiences. The explanations and the associated remindings then get incorporated into your knowledge-base of predications and help prevent a repetition of the failure. They believe that schools can take advantage of computers because they provide opportunities for children to make mistakes and fail without the stigma normally associated with negative reinforcement.

What Logo takes from AI is an environment rich in interesting things to think about and a context for thinking about why concepts or programs did or didn't do what they were supposed to do. The assumption is that the ability to articulate the processes of your own thinking will lead to improving those processes.

Logo as a Natural Environment to Learn In

An analogy that Papert has often used to describe the way children could learn in a Logo environment is the way a child learns to speak French by living in France. There is no formal curriculum and little, if any, direct teaching; the child interacts in a culture rich in things French. The child's motivation to communicate is intrinsic, and the process of learning is natural. Taylor (1980) describes Papert's approach to using computers to present children with interesting and challenging problems to solve as a way to teach "mathematical thinking that young children can intuitively master" (page 6). Schank (1984) makes a similar point when he states that "Logo's value is that it teaches children to reason without overtly trying to do so" (page 206). In a Logo-like environment the learner forgoes the often painful and tedious task of learning new

concepts or principals in the abstract, but forms those principles or concepts in the process of working on a project of his or her own design. In addition, by incorporating "objects to think with" such as the turtle, Logo helps learners to concretize the abstract by letting them "play turtle" and relate their past experience and body of knowledge to the task of programming.

Logo as a Vehicle for Skill Development

There is little justification for incorporating programming into a school's core curriculum, especially in the lower grades, if it is perceived as just equipping students with a new skill. Schank (1984) describes programming as a relatively low-level skill that may soon become automated and unnecessary. There is no evidence to indicate that there will be a great demand for programmers in the future (Soloway, 1985). Even if that evidence existed, programming would not belong in the core curriculum, but would fit better into a specialized curriculum along with other vocational and technical skills.

There seems to be a consensus in the field and in the literature that the only real justification for teaching children programming is that it promotes what are usually referred to as problem solving or higher-order thinking skills, (e.g., Bork, 1981; Gallini, 1985; Krasnor & Mitterer, 1985; Linn, 1985; Nickerson, 1982; Papert, 1980a; Pea & Kurland, 1984a, 1984b; Salomon & Perkins, 1987; Soloway, 1985, 1986). But, it's hard to teach thinking if you don't know what it means or how to do it. Unfortunately many schools implement programming to foster higher order thinking skills without a clear understanding what these skills are.

Problem Solving Skills

Presseisen (1985) describes problem solving as using basic thinking processes to:

- resolve a known or defined difficulty;
- assemble facts about the difficulty and determine additional information needed;
- infer or suggest alternate solutions and test them for appropriateness;
- potentially reduce solutions to simpler levels of explanation and eliminate discrepancies; and
- provide solution checks for generalizable value.

Gagne (1977) points out that problem solving involves more than just the application of previously learned rules. In addition to a solved problem, the process of creating and testing hypotheses results in new learning.

Higher Order Thinking Skills

Higher order thinking involves complex processes and nonroutinized thinking that differ from the standardized, competency-based instruction offered by most schools. Cuban (1984) describes the problem of defining higher order thinking as a *conceptual swamp*. He concludes that the notion of thinking skills, critical thought,

and problem solving as they exist in the minds of teachers, administrators, and parents are, for the most part, indistinguishable (pages 676-677). While there is similar disagreement in the literature on what constitutes higher order thinking, Resnick (1985) lists the following key features of higher order thinking:

- Higher order thinking is non-algorithmic. That is, the path of action is not fully specified in advance.
- Higher order thinking tends to be complex. The total path is not "visible" (mentally speaking) from any single vantage point.
- Higher order thinking often yields multiple solutions, each with costs and benefits rather than unique solutions.
- Higher order thinking involves nuanced judgment and interpretation.
- Higher order thinking involves the application of multiple criteria, which sometimes conflict with one another.
- Higher order thinking often involves uncertainty. Not everything is known that bears on the task at hand.
- Higher order thinking means self-regulation of the thinking process. We do not recognize higher order thinking in an individual when someone else "calls the plays" at every step.
- Higher order thinking involves imposing meaning and finding structure in apparent disorder (page 10).

Pogrow (1985) states that higher order thinking skills can be simply expressed as those skills that students use to analyze and synthesize information. He urges schools to go beyond helping students develop specific skills and to set a goal to help them "develop thinking ability, the ability to construct sophisticated associations between ideas" (page 27).

The assumption underlying programming as a catalyst to higher order thinking is that in the process of learning to program and in the process of programming itself, children will learn about their own thinking processes. As Papert describes it, they will be able to "articulate the workings of their own mind." The activity of programming a computer involves teaching the computer how to do something. To be successful, the programmer needs to truly understand what it is he or she is trying to teach the computer and be able to analyze the task or problem to be solved in a way that it can be "articulated" to the computer.

Mathematical Skills

The Logo philosophy also stresses that we are all mathematically minded, but that most of us lose touch with our mathematical mindedness. The belief is that people feel that they can't understand or "do" math because powerful mathematical ideas are not well represented in the school or general cultures. Papert (1980a) describes school mathematics as "denatured," "devitalized," and "depersonalized" because math is not connected to things that children think about, care about, or relate to. Logo programming is seen as a way of getting children involved in mathematics in an active and

personal way. Hoyles (1985) sees the underlying philosophy of Logo as being a catalyst for provoking mathematics educators "to reconsider what is appropriate in the mathematics curriculum in terms of conceptual content, ways of representing content, ways of drawing together cross-curricular experiences (from areas such as art, craft, and science), and finally, organization and methods in the classroom" (page 23).

So from a Logo perspective, the emphasis of programming should be on the high-level goals that will, in turn, lead to the development of powerful intellectual skills. These skills will be developed after the child invents and debugs many programs over a long period of time. This experience will lead to the development of general heuristics (what Papert calls "powerful ideas") that can be applied not only to computer programming, but to all kinds of procedural knowledge. Groen (1981) believes that these powerful ideas are at the heart of Piagetian theory because they reflect very high level logico-mathematical structures that "are responsible, in the long run, for higher levels of intelligent human activity" (page 307). Again, the assumption is that these skills will be transferable to a much broader range of problems and situations than just programming in Logo. This implies that students will need some mastery of the low level skills of program ming before they can begin to develop the higher level skills gained through the process of programming.

Logo as an Exploratory Environment

Psotka (1985) describes the Logo turtle as a device that "encourages children and other users to imagine themselves in the computer environment and to bring their knowledge and experience of the world into that environment" (page 5). Solomon (1986) points out that Logo was developed to host any number of rich computational environments. In addition to the graphic and geometric world of the turtle, Logo "microworlds" have been developed to explore areas such as music, language, biology, and physics. Papert (1980b) claims that these microworlds are implementations of Piaget's theory because children can learn by themselves without the direct intervention of educators. Logo explorations are seen as play and are initiated by the child for his or her own enjoyment.

This interpretation of Piaget holds that children are the builders of their own intellectual structures. In order to build these structures, children need access to appropriate materials, and the culture or environment they live in is the source of these materials. In Papert's (1980b) view "when the culture is rich in relevant materials they (children) build well, stably, and early. When the culture is poor in materials the building is impeded" (page 994). This perspective on intellectual development places a major emphasis on the child's environment and how he or she interacts in that environment. Developmental differences are seen as a factor of the availability of materials in the environment that, the use of which, will lead to the development of more advanced intellectual structures. Logo was designed to be an environment rich in the materials upon which these structures could be built. Since the environment is rich in materials (content), the child's explorations determine the curriculum.

Through exploring the rich microworlds of Logo, Papert (1980a) claims that

children will establish "an intimate contact with some of the deepest ideas from science, from mathematics, and from the art of intellectual model building" (page 5). This intimate contact is established through the child's ability to try out/play with the underlying powerful ideas that exist in the Logo microworlds in concrete ways. The computer provides a medium for children to concretely experience and build on knowledge that, without the computer, is accessible only through formal processes. As Groen (1984) states, "what the child is learning when involved in (a Logo) microworld is not a programming language but a way of establishing correspondences between a concrete world and one of abstract representations" (page 50).

Logo as an Aid to Reflection

A major benefit of Logo is the positive effect that it is thought to have on reflective thinking or metacognitive skills. This belief is derived from the Piagetian notion that higher levels of thought are associated with the ability to view and analyze your behavior and thought abstractly (e.g., Inhelder & Piaget, 1969). Presseisen (1986) describes metacognitive skills as "the learning to learn skills aimed at making thinking more conscious and the student more aware of the ways that one can go about problem solving or decision making" (page 9). Metacognition involves students in becoming conscious of their own ways of thinking, of exploring various possible routes to resolving a particular problem, and in becoming intellectually engaged with the learning experience through self-motivation and curiosity. Presseisen notes that research indicates that metacognitive skills need to be directly taught. Without some kind of intentional instruction there is doubt about whether metacognitive skills will result as an indirect outcome of classroom learning. At the same time, metacognition requires that the teacher must strive for student autonomy.

Clements (1986) believes that it is likely that metacognitive experiences occur during Logo programming since "children consciously solve self-selected problems using unfamiliar strategies. They must communicate their organization of the task and the solution processes to each other, to the teacher, and to the machine. They must analyze their own thought processes for errors frequently and have adequate time to do so" (page 310). Groen (1981) believes that, based on Piagetian theory, the child who has created a computer program must have, in his or her head, some structure that models the functioning of the program. As the child progresses through the debugging process, the structure needs to change in the same way that the program changes. It follows that the child's structures will accommodate and assimilate reality as a result of the debugging process. In addition, the child will be more likely to be aware of and have control over this process of accommodation and assimilation.

In addition to Piaget, Papert (1971) also credits the work of John Dewey and Maria Montessori in basing the Logo philosophy on the belief that children learn best by doing and by thinking about what they do. He believes that educational innovation must consist of better things to do and better ways to think about oneself doing these things. From this perspective, the intellectual growth of a child must be rooted in his or her experience. Logo is designed as an environment that will provide the kind of

experiences that will nurture the growth of intuitions and concepts that children will encounter in the course of thinking, learning, and playing.

When children program a computer to do what they want it to do, they need to understand how they would go about doing the task themselves. With Logo, the value of programming is that it helps children become conscious of their own thinking processes. Weir (1987) believes that "conscious self-reflection is slow to develop, and we mature as individuals to the extent that we can look at our own functioning. It must be of benefit to a growing child to be encouraged by the very nature of the learning environment to 'look at her own thinking'" (page 79). The computer programs that children create become concrete representations of their own thinking. Logo graphics provide immediate visual feedback where bugs stand out and can be fixed in a concrete way. It becomes easier to examine your own thoughts after they have been externalized as the "thoughts" of the turtle. This allows you to look at "the way the computer achieves a task as a way of thinking about how you would do it. One can use this process as part of introducing the student to the habit of 'reflective consciousness'—the ability to direct her awareness to her own thoughts" (Weir, 1987, page 81).

By reading and watching their programs run, the children/programmers can stand back and actually watch themselves solve a problem. Self-reflection is enhanced because it is easier to move back and forth from the concrete to the abstract. In assessing the value of this type of reflective learning, Clements (1985) states that "the Logo programming environment holds the promise of being an effective device for cognitive process instruction—teaching how rather than what to think" (page 63).

As noted earlier, Logo takes from AI the theory that people learn from their mistakes. In programming, finding, and fixing mistakes is known as debugging. Clements (1986) describes how debugging has been hypothesized "to be central to all learning" (page 309). The theory is that the process of finding and correcting errors forces us to focus on important misconceptions, and that when faced with similar problems in the future we will have corrected our "faulty thinking."

When children (or adults) program a computer they rarely get their programs "right" the first time. Unlike so many other experiences that children have, not being right while they are programming isn't necessarily bad, but just indicates a "bug" that needs to be fixed. They begin to understand that a program will usually require some "refinement" before it is finished and the computer makes it easy for them to fix their mistakes. Because debugging is such an integral and natural part of the programming process, it is seen as an ideal vehicle for getting children to engage and believe in this type of self-reflective thinking.

Programming requires children to be conscious of, and focus on, their thinking processes in both developing and refining their solutions to programming problems. This makes programming a promising medium for fostering the development of cognitive and metacognitive skills.

A Definition

Logo is a high-level computer programming language that has its roots in the constructivist theories of Jean Piaget, and in the need to better understand our own thought processes taken from artificial intelligence. It was designed to provide children with an environment that is rich in interesting things to do and to think about. In the process of learning and using Logo, children are thought to develop higher order logical/mathematical skills that are applicable to a much broader range of tasks than just programming. These skills are developed in the course of a child's intrinsically motivated explorations of Logo microworlds. These explorations require the child/programmer to engage in self-reflective behaviors that will lead to a better understanding of, and control over, their own thought processes.

Conclusion

Logo has been hampered by the lack of a commonly accepted definition. This has led, in many cases, to implementations with unclear or contradictory goals and fuzzy methodologies. Interpretations range from a delivery system requiring a hands-off approach, to a lock-step progression through the syntax and semantics of the language. Regardless of the interpretation, there is almost always an assumption that higher order skills will result. Usually, the result is disappointment.

Logo is a complex innovation that encompasses physical, social, pedagogical, and curricular change. Along with progressive education, open classrooms, team teaching, and a host of other complex educational innovations, the developers of Logo have not been able to communicate effectively to the schools exactly what it is all about and how to use it (see, for example: Baldridge, 1975; Havelock, 1971). One reason for this difficulty is an educational system with an inclination to look for simple solutions or quick fixes and that lacks the desire or resources to undergo extensive reform.

Another reason for the difficulty in effectively defining educational innovations is the gulf between the developers and practitioners. Cohen (1987) points out that the intellectual inspiration for most educational innovations almost always comes from elite institutions, but that "prestige does not easily translate into influence on practice" (page 166). The knowledge produced at research centers is often seen as irrelevant or inaccessible to teachers in the field and also to the faculty of the institutions where they are educated.

Translating Logo theory into practice will require developing more effective ways of communicating what the theory is and how to practice it.

References

Baldridge, J. V. (1975). Organizational innovation: Individual, structural, and environmental impacts. In J. V. Baldridge & T. E. Deal (Eds.), *Managing change in educational organizations* (pages 151-175). Berkeley: McCutchan.

Bork, A. (1981). *Learning with computers.* Bedford, MA: Digital Books.

Case, R. (1978). Intellectual development from birth to adulthood: A neo-Piagetian interpretation. In R. S. Siegler (Ed.), *Children's thinking: What develops?*

Hillsdale, NJ: Lawrence Erlbaum Associates.

Clements, D. H. (1985). Research on Logo in education: Is the turtle slow but steady, or even in the race? *Computers in the Schools, 2*, 55-71.

Clements, D. H. (1986). Effects of Logo and CAI environments on cognition and creativity. *Journal of Educational Psychology, 78*, 309-318.

Cohen, D. K. (1987). Educational technology, policy, and practice. *Educational Evaluation and Policy Analysis, 9*, 153-170.

Cuban, L. (1984). Policy and research dilemmas in the teaching of reasoning: Unplanned designs. *Review of Educational Research, 54*, 655-681.

Cuban, L. (1986). *Teachers and machines: The classroom use of technology since 1920*. New York: Teachers College Press.

Favell, J. H. (1963). *The developmental psychology of Jean Piaget*. Princeton, NJ: Van Nostrand.

Feuerstein, R. (1980). *Instructional enrichment: An intervention program for cognitive modifiability*. Baltimore: University Park Press.

Gagne, R. M. (1977). *The conditions of learning* (3rd ed.). New York: Holt, Rinehart, and Winston.

Gallini, J. K. (1985). Instructional conditions for computer-based problem-solving environments. *Eductional Technology, 25*(2), 7-11.

Groen, G. J. (1981). The theoretical ideas of Piaget and educational practice. In P. Suppes (Ed.), *Impact of research on education: Some case studies*. Washington, D.C.: National Academy of Education.

Groen, G. J. (1984). Theories of Logo. *Pre-Proceedings of the 1984 National Logo Conference*, 49-54.

Havelock, R. (1971). *Planning for innovation through dissemination and utilization of knowledge*. Ann Arbor: University of Michigan.

Harvey, B. (1985). *Computer science Logo style: Intermediate programming*. Cambridge, MA: The MIT Press.

Hoyles, C. (1985). Developing a context for Logo in school mathematics. *Logo 85 Theoretical Papers, 1*, 23-42.

Inhelder, B., & Piaget, J. (1969). In E. A. Lunzer & D. Papert (Trans.), *The early growth of logic in the child: Classification and seriation*. New York: Norton.

Krasnor, L. R., & Mitterer, J. O. (1985). Logo and the development of general problem-solving skills. *The Alberta Journal of Educational Research, 30*, 133-144.

Linn, M. C. (1985). The cognitive consequences of programming instruction in the classroom. *Educational Researcher, 14*(5), 14-16; 25-29.

Nickerson, R. S. (1982). Computer programming as a vehicle for teaching thinking skills. *Thinking, 4*, 42-48.

Papert, S. (1971). *Teaching children thinking*. (LOGO Memo No. 2). Cambridge: MIT, Artificial Intelligence Laboratory.

Papert, S. (1980a). *Mindstorms: Children, computers and powerful ideas*. New York: Basic Books.

Papert, S. (1980b). Redefining childhood: The computer presence as an experiment

in developmental psychology. In *Proceedings IFIP*, (pages 993-998). Amsterdam: North-Holland.

Pea, R. D., & Kurland, D. M. (1984a). *Logo programming and the development of planning skills* (Tech. Rep. No. 16). New York: Bank Street College of Education, Center for Children and Technology.

Pea, R. D., & Kurland, D. M. (1984b). On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2*, 137-168.

Piaget, J. (1972). Intellectual evolution from adolescence to adulthood. *Human Development, 15*, 1-12

Pogrow, S. (1985, April). Helping students to become thinkers. *Electronic Learning*, pages 26-29, 79.

Presseisen, B. Z. (1985). Thinking skills: Meanings and models. In A. L. Costa (Ed.), *Developing minds: A resource book for teaching thinking*. Alexandria, VA: Association for Supervision and Curriculum Development.

Presseisen, B. Z. (1986). *Thinking skills: Research and practice*. Washington, D.C.: National Education Association.

Psotka, J. (1985). *Reflections on computers and metacognition*. Alexandria, VA: Army Research Institute. (ERIC Document Reproduction Service No. ED 263 206)

Rich, E. (1982). *Artificial Intelligence*. New York: McGraw-Hill.

Resnick, L. B. (1985). *Education and learning to think*. Pittsburgh: Learning Research and Development Center, University of Pittsburgh.

Salomon, G., & Perkins D. N. (1987). Transfer of cognitive skills from programming: When and how? *Journal of Educational Computing Research, 3*(2), 149-169.

Schank, R. C. (1984). *The cognitive computer*. Reading, MA: Addison-Wesley.

Schank, R. C., & Slade, S. (1985). *Education and computers: An AI perspective* (Tech. Rep. No. RR 431). New Haven: Yale University, Department of Computer Science.

Solomon, C. (1986). *Computer environments for children*. Cambridge, MA: The MIT Press.

Soloway, E. (1985). *Why kids should learn to program* (Research Report #420) Yale University, Computer Science Department.

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM, 29*, 850-858.

Taylor, R. P. (Ed.). (1980). *The computer in the school: Tutor, tool, tutee*. New York: Teachers College Press.

Weir, S. (1987). *Cultivating minds: A Logo casebook*. New York: Harper & Row.

Winston, P. H. (1977). *Artificial intelligence*. Reading, MA: Addison-Wesley.

Abstracted Knowledge: A Mid-Road Transfer Approach to Critical Thinking

Clifton S. Harris
The University of Texas at Dallas

Clifton S. Harris is an associate professor in the School of Human Development at the University of Texas at Dallas. His research interests include human-machine interaction with applications to education.

Introduction

Thinking is a generic skill needed across all disciplines. It has a long history of advocacy, ranging from Aristotle to Sir Francis Bacon to John Dewey. These luminaries are now joined by a host of new investigators who are pursuing the topic of critical thinking. A recent convergence has helped to focus this surge of interest: the advent of the Information Era and some of its insights coupled with the shift of priority attention from worldwide political competition to economic competition to educational competitiveness. The latter has brought renewed concern for an excellence in academic achievement that includes a concerted emphasis on the teaching of higher order thinking. In fact, it is reasonable to say that there is unprecedented interest in critical thinking as a fundamental skill.

Content-oriented vs. Content-free Views

Unfortunately, past attempts at teaching critical thinking do not transfer as smoothly and effectively as had been hoped. For instance, strictly content-oriented efforts to teach thinking have had supporters in researchers such as R. Glaser (1984) and Resnick (1983). Also, this view has gleaned some support from schema theory and a strong following in the cognitive psychology community (Brewster & Nakamura, 1984). However, schemas appear to inhibit transfer when a learner tries to jump from one schema to another. Similarly, abstract frames have strong proponents in the artificial intelligence community (Minsky, 1975), but their proclivity toward predefined slots in memory work against transfer. Additionally, E.M. Glaser (1985) maintains that there is little evidence that students acquire skill in critical thinking as a necessary byproduct of the study of any given subject. Although domain-specific knowledge is essential to good thinking within a domain, critics claim such thinking is not sufficient to assure that good thinking will occur in other course experiences (Nickerson, 1985). Transfer gains appear to be limited to tests that are highly similar in content and structure.

On the other side, there are proponents of general thinking skill courses that have positive effects that are transferable to a variety of content-specific domains (Halpern,

1984; Block, 1985). This approach seems predicated on two basic assumptions: (1) that there are clearly identifiable and definable thinking skills that students can be taught to recognize and apply appropriately, and (2) that if these skills are recognized and applied, the students will be more effective thinkers. Such skills ordinarily include recognizing and critiquing stated and unstated assumptions, articulating correct inferences, analyzing means-ends relationships, reducing complex problems to sub-problems, and others. Adams (1989) claims (from a cognitive point of view) that this approach and the related content-free (abstract) efforts have not shown desired promise. McPeck (1981) argues (from an epistemological approach) that critical thinking is necessarily subject-specific and is not generalizable.

Thus, research efforts and literature debates continue about the merits of levels of processes and principles (micrological vs. macrological skills) and the nature of materials (abstract vs. real world or ecologically valid materials). Micrological skills embrace abstract materials such as simple geometric shapes, dot matrices, and elementary skills such as observation, classification, and seriation. (Such skills are identified with the content-free view.) Macrological skills refer to ability to cope with real world problems that tend to be complex rather than classroom contrivances, and creativity that enables one to synthesize and compose in the midst of multiple alternatives and options. (Such skills tend to be identified with the content-oriented view.) Lack of agreement and controversy abounds among content-oriented and content-free advocates. Perhaps the resolution lies in some middle ground between the polar positions.

Abstracted Knowledge as Middle Ground

Knowledge becomes abstracted through the deliberate decontextualization of a principle, main idea, strategy, or procedure that then becomes a candidate for transfer (Perkins & Salomon, 1989). Understandably, metacognitive guidance plays a major role in a process of this nature. For example, a teacher of history could expose a student to the major events in American history during the 1700-1800 period and then guide the students to decontextualize the cycle of oppression-revolution-independence. Such guidance may lead the student to see that the cycle in American history transfers to the French and Mexican experience during the same general time frame and to other historical periods as well. These authors argue that the payoff of such activities is not just particular transfers but the establishment of an expectation for transfer. The expectation of transfer alerts the student to the process of deliberate abstraction. It is important that students learn that such connection making is vital to thinking historically, literarily, and scientifically. Further, it is fundamental to this notion that critical thinking skills become abstracted through their repeated exercises in a diversity of contexts and problem situations. The diversity of contexts enables students to comprehend, expect, and exercise transfer across subject domains. In time, as the particular instances of application overwrite each other, their distinctive features give way to their commonalities (Adams, 1989).

Emerging evidence of the potential effectiveness of a middle ground can be found in Lipman's *Philosophy for Children* program (Lipman, 1985) that tends to be a hybrid on the micro-macro continuum. Additional promise for a "mediated" transfer approach comes from a developmental project known as *Odyssey: A Curriculum for Thinking* (Adams, 1986). Striving for the best of the micro and macro approaches, this program emphasizes abstract knowledge that is conceptually neutral and incorporates material that is supposedly "equally generalizable" to all applicable problem domains. This author believes that a promising case can be made for judicious use of abstracted knowledge approaches in which content is chosen from a diversity of contexts and problem situations. It has the advantage of relying upon high-road transfer (Perkins, 1987), which occurs by way of mindful abstraction from the context of learning for application to another context. It demands the conscious effort of the learner in seeking generalizations and applications beyond the obvious. Mindfulness may be cultivated by cueing students to try to apply information recently learned. In effect, we are establishing a pattern of inducing a search through recently obtained information for the purpose of making relevant connections. This writer's experience in teaching critical thinking courses argues for thinking skills to be mediated explicitly, rather than implicitly, in order for students to perceive the applicability of critical thoughts and ideas. Such argument is consistent with Schoenfeld's (1975) research demonstrating the efficacy of explicit mediation. In other words, this approach accomplishes an important "bridging" function, which has been ably articulated by Feuerstein (1980).

Indirect Support for the Middle Ground

Indirect support for the mindful abstraction view is gained from Salomon (1988) and his concept of cognitive residue. He uses the vehicle of intelligent computer tools to foster the notion that such tools affect cognitions that can later be used without the scaffolding of external, intelligent tools. Such tools provide a guidance and modeling function that lead to internalization of a tool's strategies or modes of representation such that it serves as a cognitively reconstructed tool. Some conditions for tool internalization are (1) potentially they could be carried out in one's mind, (2) they should be of a generalizable nature, potentially applicable to new instances, (3) they should serve a novel and useful cognitive function, and (4) they should be explicit so that a child can actually witness, trace, and reproduce them.

Indirect support is not limited to cognitivists. For instance, Siegel (1988), an epistemologist, argues that a critical thinker must have a well-developed "disposition" to engage in reason assessment. He says that one who has a critical attitude necessarily has a certain character as well as certain skills—a character that is inclined to seek and to have judgment and action based on reasons. Or, to state his view in psycho-social terms, one might say that students' self-perception of their ability to engage in mindful abstraction may be a more critical influence on thinking than actual personal skill or task incentive.

Integrative Perspective for the Middle Ground

As a minimum, this argument should be anchored in the insights on transfer provided by contemporary cognitive science, and it should have a well-integrated schema to guide its implementation. A practical consideration is how should the effort be implemented? Should thinking skills be separated into stand-alone courses or be infused into existing curricula? Each type of program has important advantages. This author agrees with Sternberg (1987) that the arguments mounted on both sides are persuasive enough to advocate a mixed model in which thinking skills are taught as a separate course at the same time they are infused throughout the entire curriculum. The challenge of teaching such skills are so imposing that any culture needs the impact of both approaches.

Next, the assessment issue deserves comment because of the unique difficulties presented. One of the most promising means of exploring course related changes in thinking ability is to examine the underlying structure of cognitive skills and knowledge based on comparisons between skilled and unskilled problem solvers. Such an approach could attempt to demonstrate that novices who are given instruction or schemata are more similar to experts than a control group of novices who had some other educational experience (Halpern, 1987).

Since transfer is not a unitary phenomenon, the mid-ground position of abstracted knowledge may well contribute to a more effective means of cultivating mindful, insightful, critical thinking. It may advance the frontiers of thinking to the extent that the vision of philosophers such as Siegel (1988) may be realized. He believes that critical thinking may emerge as a fundamental educational ideal, infusing and informing the entire range of educational activities and affairs.

References

Adams, M.J. (Ed.) (1986). *Odyssey: A curriculum for thinking* (Vols. 1-6). Watertown, MA: Charlesbridge Publishing.

Adams, M.J. (1989). Thinking skills curricula: Their promise and progress. *Educational Psychologist, 24*(1), 25-77.

Block, R.A. (1985). Education and thinking skills reconsidered. *American Psychologist, 40*, 574-575.

Brewster, W. E., & Nakamura, G.V. (1984). The nature and function of schemas. In R.S. Wyer & T.K. Srull (Eds.), *Handbook of social cognition* (pages 119-160). Hillsdale, NJ: Lawrence Erlbaum Associates.

Feuerstein, R. (1980). *Instrumental enrichment: An intervention program for cognitive modifiability*. Baltimore, MD: University Park Press.

Glaser, R. (1984). Education and thinking: The role of knowledge. *American Psychologist, 39*, 93-104.

Glaser, E.M. (1985). Critical thinking: Educating for responsible citizenship in a democracy. *National Forum, 65*, 24-27.

Halpern, D.F. (1984). *Thought and knowledge: An introduction to critical thinking*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Halpern, D.F. (1987). Analogies as a critical thinking skill. In D.E. Berger, K. Pezdek, & W.P. Banks (Eds.), *Applications of cognitive psychology: Problem solving, education, and computing*. Hillsdale NJ: Lawrence Erlbaum Associates.

Lipman, M. (1985). Thinking skills fostered by philosophy for children. In J.W. Segal, S.F. Chipman, & R. Glaser (Eds.), *Thinking and learning skills, Vol. 1: Relating instruction to research* (pages 83-108). Hillsdale, NJ: Lawrence Erlbaum Associates.

Minsky, M. (1975). A framework for representing knowledge. In P.A. Winston (Ed.) *The psychology of computer vision* (pages 211-227). New York: McGraw-Hill.

Nickerson, R., Perkins, D., & Smith E. (1985). *The teaching of thinking*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Perkins, D. (1987). Thinking frames: An integrative perspective on teaching cognitive skills. In J.B. Baron & R.J. Sternberg (Eds.), *Teaching thinking skills: Theory and practice*. New York: W.H. Freeman and Co.

Resnick, L. (1983). Mathematics and science learning: A new conception. *Science, 220*, 477-478.

Salomon, G. (1988, April). *AI in reverse: Computer tools that become cognitive*. Paper presented at the Annual Meeting of the American Educational Research Association, New Orleans.

Salomon, G., & Perkins, D. (1989). Rocky roads to transfer: Rethinking mechanisms of a neglected phenomenon. *Educational Psychologist, 24*(2), 113-142.

Schoenfeld, A. (1979). Explicit heuristic training as a variable in problem solving performance. *Journal for Research in Mathematics Education, 10*, 173-187.

Siegel, B. (1988). *Educating reason—rationality, critical thinking, and education*. New York: Routledge.

Sternberg, R. (1987). Questions and answers about the nature and teaching of thinking skills. In J.B. Baron & R. J. Sternberg (Eds.) *Teaching thinking skills: Theory and Practice*. New York: W. B. Freeman and Co.

Resolving the Impasse in Software Engineering: Problem Solving in Program Construction

Warren Moseley
Texas Instruments, University of Alabama in Huntsville

Warren Mosley is a resident-affiliate of Texas Instruments to the Software Engineering Institute at Carnegie Mellon University. His research interests include software engineering, artificial intelligence, intelligent tutoring systems, and computer music.

Introduction

A difficult part of software engineering to study theoretically or empirically is the cognitive aspect of knowledge engineering. Guindon and Curtis (1988) suggest that a software methodology built on a model of learning makes the software development process simpler, and, in turn, allows the software developer to work on more significantly complex domains.

"Getting stuck" while problem solving is called an impasse. VanLehn (1988) explains that this impasse occurs when the next step cannot be performed. Inductive learning occurs at this impasse when the student takes the corrective action to repair the impasse. This paper accents the analogy that a software engineer is a combination of student and teacher. Help obtained from a software support environment by the student (or designer) reduces to a sequence of actions to repair the impasse. This generated action sequence is then abstracted by the student and is inserted into the old procedure at exactly the location where the impasse occurred.

In experimentation with the concept of impasse-driven software engineering the monitoring of the user interface becomes a matter of increasing importance, particularly with respect to the derivation of an impasse that must be repaired. Creating a support environment for impasse-driven software engineering makes it necessary to classify the user interface domains as:

D1—the subset of concepts (and their associated commands) that the users know and use without noticeable problems.

D2—the subset of concepts that are used occasionally. Users usually do not know the details about them, and are not certain about their effects. In this category of concepts are the commands that are summarized by help cards, keyboard overlays, on-line hints, and safeguards. An example of the use of a safeguard would be an undo command. Gradually concepts in D2 category migrate to D1.

D3—the subset of concepts the user thinks exist in the system. A passive help facility is necessary for the user to communicate his or her plans to the system.

D4—this category represents all of the features of the system. Passive helps are

of little use in this domain because the user seldom knows of the existence of these commands or features. Active help agents guide a user to an intelligent colleague or assistant so that the user can incrementally extend his or her knowledge.

The D4 domain requires the assistance of a coach or guide, where as domain D3 allows freedom to browse and explore. The system model of the user will be used to determine the domains D1 to D4 and how abstractions are utilized in this domain analysis. The classification of actions monitored in the user interface will be used to map actions into one of the above domain classifications. Once the domain is identified, then the software support monitor will attempt the help the user determine when an impasse occurs. To understand this impasse detection, it is necessary to understand the rationale for the construction of an abstraction.

The Importance of Abstraction

John Locke in a 1690 work "An Essay on Human Understanding" states, "The acts of the mind, wherein it exerts power over simple ideas, are chiefly these three:

1. Combining several ideas into one compound one, and thus all complex ideas are made.
2. The second is bringing two ideas, whether simple or complex, together, and setting them by one another so as to take a view of them at once, without uniting them into one, by which man gets all its ideas of relations.
3. The third is separating them from all other ideas that accompany them in real existence: this is called **abstraction**, and thus all its **general ideas** are made."

A difficult concept to understand and to explain is how an abstraction is formed. Large systems are difficult to develop because they are conceptually complex, and it is difficult to determine what abstractions are necessary and how these abstractions will be linked together. Increased complexity demands tool support to assist in the management of the overall complexity. Luqi and Berzinis (1990) suggest that more formal approaches to the building of specifications are valuable because they: 1) support formal reasoning and automated processing, 2) explicitly record agreements made among the customers, designers, and programmers on the system under construction, 3) help the designer understand and use abstract objects independently of particular implementations, 4) protect programmers working at one level from the implementation details and one another, and 5) insulate users from irrelevant implementation concepts. Furthermore, computer-assisted tools that detect design errors require more formal specifications.

Abelson and Sussman (1985) suggest four fundamental issues regarding problem solving. The first three of these issues have to do with language expression. These issues are:

1. **Primitives**—the ability to express things in a form that cannot be decomposed.
2. **Means of Combination**—the ability to form compound expressions.

3. **Means of Abstractions**—the means by which simple and compound objects are named and manipulated as units.

When the understanding of the internal mechanics of an algorithm to compute square root is no longer necessary, then the problem solver has been successful at defining an abstraction barrier. The significance of the establishment of abstraction barriers is that once this abstraction barrier is constructed, then the abstraction becomes a primitive in the language and can then be combined to form higher abstraction layers. A serious question is raised, "What dictates when an abstraction is necessary, and how can one determine that an abstraction should be introduced into the software product?"

Sussman and Abelson suggest that the fourth and perhaps the most difficult problem-solving issue is understanding the common patterns of usage in a problem domain. This issue is closely related to the concept of an impasse. If one could accurately describe a process, then an impasse would not occur because the common pattern of usage is identified and applied. As a preliminary means of theory verification, the process of unsticking this "getting stuck" approach has been mapped from theory to an empirical study of the program structuring process.

Other objectives of the study of problem solving in program construction are:

1. To study the Incremental Learning in the context of the software development process, and to introduce into the software support environment tools and techniques to capture the essence of an Incremental Learning Process.
2. To understand **where** in the software development process an **impasse exists**, and to capitalize on that impasse to help the software engineer focus cognitive skills on the proper part of the solution.
3. To understand the role of the software engineer as a combination student/teacher, and to apply known principles of learning, teaching, problem solving, and technology transfer.

Software Engineering is a Learning/Teaching Process

This section will examine various principles used by software engineers and will compare these principles to the learning process. The learning/teaching approach to software design encourages the engineer to improve his/her understanding of the problem domain, which includes making predictions about exhibited phenomenon, formulating experiments to validate the predictions, assessing the associated risks of the observation, and choosing a final selection on the correct representation of the problem domain. This learning approach is then turned into a teaching component as a means of communicating knowledge. This conceptual transfer process is usually done through the software specification. Hence, software engineering can be viewed as a knowledge communication process contingent upon the prerequisite knowledge compilation process.

Heuristics Used by Software Engineers

It is important to understand heuristics used by software engineers when learning about a new problem domain and improving the knowledge available for solving a particular problem within that domain. Consider the example of drawing a square using turtle graphics (Abelson, 1986). In turtle graphics the imaginary turtle is given commands to move and turn, and all graphics operations are done in this context. The following example would produce a square.

```
TO SQUARE
   FORWARD 100
   RIGHT 90
   FORWARD 100
   RIGHT 90
   FORWARD 100
   RIGHT 90
   FORWARD 100
```

This is an example of an abstraction. Once the procedure SQUARE is named, then the SQUARE can be used to create more complex objects. In this case the user only has to know that the turtle will start at a given point, move forward a certain number of units, and turn right until the square is complete. The fact that it takes 4 turns to complete this figure might not be known at the beginning of the task. It might take D3 (discovery by doing) domain knowledge to realize that it takes at least three right turns to complete the entire square. The size of the turn is quite straightforward in this case. The turn would be 90 degrees for any 4-sided polygon.

Consider the following two examples. In normal geometric figures the angles are measured in terms of the internal angle, such as in Figure 1B. However in turtle commands, **Right N** causes the turtle to reorient the heading by the number of degrees of the turn. The angle is actually the outside angle, not the inside angle. In this example it appeared quite logical to use 60 degree turns until Figure 1A is created and the problem is identified. This is an example of a simple impasse. Until the user makes the design repair, the triangle will not produce the figure in Figure 1B. The repair is to substitute the value of the external angle rather than the internal angle.

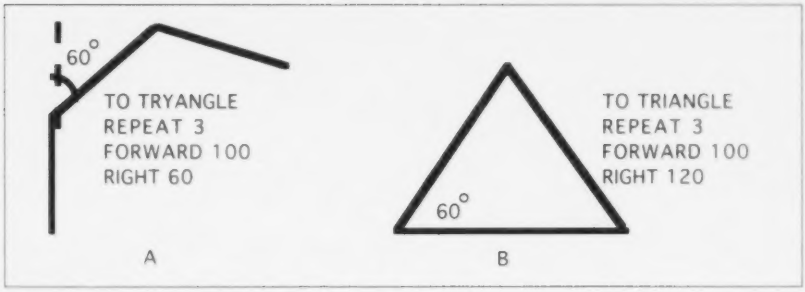

Figure 1. Creating a Triangle

Consider a more complex example. A difficult concept to teach in any programming science class is the common pattern of recursion. The turtle graphics representation can be used to demonstrate this problem using a regular binary tree. In this example a branch will sprout two branches, and the length of the new branch will be one half the length of the previous branch. Conceptually each branch consists of a straight line, with two more branches at the end. Notice the fundamental recursive nature of the definition. A branch consists of a stem plus sub-branches (things with the same structure as the branch). Consider the following procedure.

```
TO BRANCH LENGTH
   (Draw the Main Branch)
   FORWARD LENGTH
   (Turn to point along left secondary branch and draw it.)
   LEFT 45
   BRANCH LENGTH/2
   (Now draw the right branch)
   RIGHT 90
   BRANCH LENGTH/2
```

The procedure BRANCH produces the example in Figure 2. The tree would continually generate left branches and never return to draw a right branch. This is a fundamental problem in teaching recursion and represents a basic misunderstanding of the concept of recursion and the use of this concept to draw a tree.

Figure 2. First Attempt to Draw a Tree

This first impasse occurs and the usual repair is to insert the following procedural change (shown in bold type). The variable level provides a stopping point for the recursive nature of the tree.

```
TO BRANCH (LENGTH, LEVEL)
   IF LEVEL=0 THEN RETURN
          {Draw the Main Branch}
          FORWARD LENGTH
   Turn to point along left secondary branch and draw it.}
          LEFT 45
          BRANCH (LENGTH/2, LEVEL-1)
          {Now draw the right branch}
          RIGHT 90
          BRANCH (LENGTH/2, LEVEL-1)
```

Unfortunately, this does not solve the problem, and creates a new impasse. Figure 3 is the result of executing the above procedure with a level of 3.

Figure 3. Still an Impasse

The bug is tricky to find. The drawing of the secondary branch must occur after the turtle returns to the base of the branch, or the RIGHT 90 will not align properly to draw the right branch of the tree. The process that draws the left branch must restore the state of the turtle to what the state was before the process began. BRANCH will only work if the sub-branches are state transparent. However, since BRANCH is recursive, BRANCH itself must be made state transparent.

```
TO BRANCH (LENGTH, LEVEL)
   IF LEVEL=0 THEN RETURN
          {Draw the Main Branch}
          FORWARD LENGTH
   {Turn to point along left secondary branch
                and draw it.}
          LEFT 45
          BRANCH (LENGTH/2, LEVEL-1)
```

```
RIGHT 90
BRANCH (LENGTH/2, LEVEL-1)
{Turn around and back up to make the state transparent}
LEFT 45
BACK LENGTH
```

The above procedure produces the expected result.

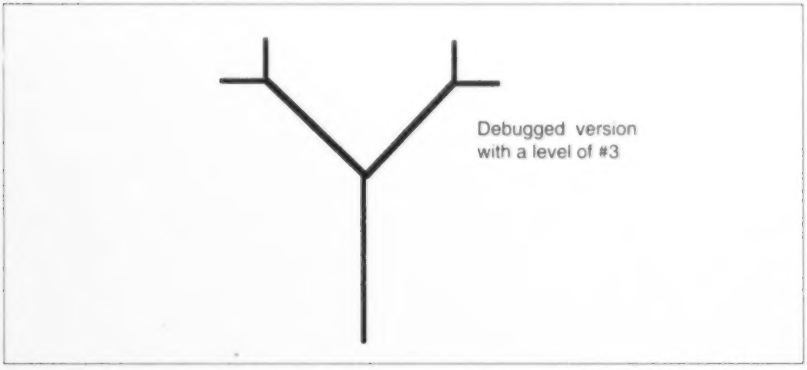

Figure 4. Nice Tree

A specific tree can be envisioned as a map of the space of the world and the relationships of the objects in that space, for which that tree is the representation. Therefore, a software engineer can be viewed as a special kind of student, one who learns about the relationships of the objects manipulated, and then organizes this learned knowledge into a representation of the space of the world, for example, the problem domain.

Impasse-driven Software Engineering

Software engineering is widely viewed as a knowledge communication process coupled with a knowledge compilation process. The communication process interprets instructions, thereby incorporating new information from the environment into mental structures of the student and into the subject domain. During the early phases of the design process, when requirements and specifications are vague, initial design control strategies are based upon a cognitive model of both knowledge sources and process sources. Guindon and Curtis (1985) have examined software designers to find that the design process is partially controlled by recognition of partial solutions at different levels of abstraction. This is considered as an opportunistic design process and contains only a minimum number of breakdowns which include:

- Difficulty in merging partial solutions into a complete solution
- Difficulty in remembering to return to postponed subproblems

Software tools should be able to help alleviate these breakdowns.

Learning by Asking Questions

A basis for problem-domain understanding in software engineering can be viewed as the ability to ask proper questions to the proper person at the proper time, and then to translate the question-answer sequences into a base of knowledge that must be communicated to the persons that will subsequently implement the software system. There is no laundry list of questions, no stereotyped sequences of asking questions, and no cookbook approach to the question-answering process. However, it is important to understand that regardless of the question sequences, it will ultimately be the robustness of the questions, the quality of the filter applied to the answers, and the ability to structure the question-answer sequences into a software specification that will be of premier importance to the generation of robust and accurate software specification.

Question-answer sequences provide a means of evaluating current user thought patterns and mapping these patterns into the general concepts to be represented within the problem domain. These sequences offer contributions to indicate the knowledge structure needed. Some of these contributions may be relevant, while others will not be relevant to the domain. Research continues to filter the relevant from the non-relevant contributions gained from question-answer sequences.

Learning and Knowledge Acquisition

The process of learning about a new problem domain and improving the knowledge available from within that domain is, in some cases, referred to as knowledge acquisition. The classification of the software design heuristics can, therefore, be seen as a knowledge acquisition process that includes domain classification of software tasks and knowledge-level analysis. Learning is directly related to the software design process, indicating that research should be directed into formalizing the learning process as a knowledge acquisition paradigm.

Knowledge acquisition is mainly a tool used to analyze the knowledge structure associated with an expert system. It involves interviewing an expert, mostly through question-answer sequences much like the initial phase of the software design process, then refining the knowledge gained into a usable representation of the problem domain. Therefore, learning, knowledge acquisition, and software design processes are closely related to the most basic cognitive process used by man and machine.

This paper focused on the understanding of critical thinking and problem solving in software engineering. There is much research left to be done in the area: what actually defines an impasse, how incremental learning strategies operate in an intelligent environment, and how visualization can assist the knowledge engineering process. The examples demonstrated in this paper are quite simplistic, but demonstrate the overall concepts.

References

Abelson, H., & DiSessa, A. (1986). *Turtle geometry: The computer as a medium for exploring mathematics*. MIT Press.

Berzins, V. & Luqi. (1990, March). An introduction to the Specification language. *Spec.*, pp. 74-75.

Guindon, R. & Curtis, B. (1988, May). Control of cognitive processes during software design: What tools are needed. *Proceeding of the Human Factors in Computing Systems* CHI '88, Washington, DC.

Locke, J. (1690). *An essay on human understanding*. Quoted in reference #2.

Abelson, H. & Sussman, G. (1985). *Structure and interpretation of computer programs*. Cambridge, MA: MIT Press.

Mandl, H. & Lesgold, A. (Eds.). (1988). *Learning issues for intelligent tutoring systems*. New York: Springer-Verlag.

Critical Thinking and Intelligent Tutoring Systems

James T. Streib
Memphis State University

James T. Streib is an assistant professor of Computer Science at the Univeristy of Central Akansas. His research interests include critical thinking and intelligent tutoring systems.

An Overview of Critical Thinking

The phrase "critical thinking" has been used to describe many different forms of thinking. It has been used narrowly to describe the form of thinking that is used only in criticism and has been used broadly to describe thinking as a whole. Although there seems to be a number of varying ideas concerning the nature of critical thinking, there does appear to be at least some common elements among all of the varying ideas.

In a general overview of critical thinking skills, Heiman and Slomianko maintain that the basic skill involved in critical thinking is that of "engaging in an internal dialogue" (1985, page 70). The basis for this ongoing internal dialogue is the "process of generating questions" that "is most useful when it promotes active student learning" (page 20).

In a more specific examination of thinking skills, Barbara Z. Presseisen (1986) has categorized thinking into four kinds of cognitive processes: essential, higher order, metacognitive, and epistemic. Some of the skills that could be included in the essential cognitive processes are "recall, recognition, comparison, classification, inference, analogy, inductive and deductive reasoning, interpretation and evaluation" (page 9). The higher order cognitive processes are based on the essential cognitive processes, and they are further broken down into four types: problem solving, decision making, critical thinking, and creative thinking. It is here that she describes critical thinking as using basic thinking processes to analyze arguments and generate insight into particular meanings and interpretations; develop cohesive, logical reasoning patterns, and understand assumptions and biases underlying particular positions; attain a credible, concise, and convincing style of presentation.

Barry K. Beyer (1988) has also categorized critical thinking as a part of the overall process of thinking. According to Beyer, thinking can be broken down into cognition and metacognition. The major cognitive operations are further broken down into three different levels where the highest level is major thinking strategies, which includes problem solving, decision making, and conceptualization. The second highest or middle level is that of critical thinking skills, and the third and lowest level is that of micro-thinking skills. Micro-thinking skills include both the basic information processing operations derived from Bloom's taxonomy and reasoning operations that include inductive, deductive, and analogical reasoning.

Critical thinking is composed of ten operations that can be used singly or in any combination (Beyer, 1988). These operations are:

1. Distinguishing between verifiable facts and value claims
2. Distinguishing relevant from irrelevant information, claims, or reasons
3. Determining the factual accuracy of a statement
4. Determining credibility of a source
5. Identifying ambiguous claims or arguments
6. Identifying unstated assumptions
7. Detecting bias
8. Identifying logical fallacies
9. Recognizing logical inconsistencies in a line of reasoning
10. Determining the strength of an argument or claim

The above list is not meant to be a taxonomy as with the micro-thinking skills, nor is it meant to be a major strategy such as the level I thinking strategies. "An individual may engage in one or more of these, in any order, and have engaged in critical thinking" (Beyer, 1988, page 61). However, just as micro-thinking skills are a part of critical thinking and major thinking strategies, critical thinking skills are also a part of major thinking strategies.

According to Beyer, critical thinking essentially "consists of judging the authenticity, worth or accuracy of something." Further he maintains that, if there is one aspect of critical thinking that can best describe critical thinking, it would be argumentation. "Probably the most all-inclusive act of critical thinking is that of argumentation—argument making and argument analyzing" (pages 61-62). However, the type of arguments used in critical thinking should not be confused with arguments used solely in the sense of logic and deductive reasoning.

An argument, in critical thinking terms, is an assertion or claim accompanied by a line of reasoning that supports the claim and that denies any alternative claims.

Although there are differences between where Presseisen, Beyer, and others have categorized critical thinking and also differences in what they consider critical thinking, there does seem to be a number of similarities. Both Presseisen and Beyer appear to stress argumentation as one of the major aspects of critical thinking. They both also seem to categorize it as a middle-level thinking skill.

However, this is not to say that critical thinking ought to be restricted only to an intermediate level. Critical thinking seems to pervade all aspects of thinking, including the higher levels of thinking. It seems as though critical thinking should not necessarily be sandwiched between various levels, and it may be the case that critical thinking should not be categorized at all. As a result, critical thinking could possibly be a separate aspect of thinking that is set off to the side and utilized by all levels of thinking. The issue of where critical thinking should be categorized or whether critical thinking should be categorized at all is not really a concern here because the important point is that critical thinking is generally agreed upon to be a higher order thinking process.

The Differences between CAI and ITS

The most prominent form of instruction utilizing computer systems is known as Computer Assisted Instruction (CAI). The earlier versions of these systems tended to be based on Skinnerian principles and stressed methods such as programmed instruction and drill and practice. Although some systems are still based on this method of delivery, content and design have improved considerably. However, these systems can sometimes still be thought of as nothing more than "electronic page turners" (Barr, 1982). Should these forms of systems be applied to the area of critical thinking, it would seem that their performance could be less than satisfactory.

It would appear that computer-based instructional systems could have a better chance of fostering critical thinking skills if they included the principles of Artificial Intelligence (AI) and Expert Systems (ES). In particular, CAI systems incorporating the principles of AI and ES are known as Intelligent Computer Assisted Instruction (ICAI) or, in other words, Intelligent Tutoring Systems (ITS). These types of systems are not based on a static data base with a preset list of questions and answers. These systems are known as adaptive or reactive systems because the questions and answers are not preset and, instead, alter their strategies based upon the actions and reactions of students.

Unlike many of the CAI systems developed by educators, ITS systems have been developed by computer scientists as research tools to test AI ideas. Although many CAI systems are currently available for use on microcomputers, ITS systems can usually be run only on mainframe computers or dedicated minicomputer systems due to their extensive processing and storage requirements. As a result, it is difficult to acquire and test out ITS programs to determine their suitability in helping students acquire critical thinking skills.

A Review of Five ITS Systems

In spite of the above limitations, this paper will examine the literature of five ITS systems in an attempt to determine their suitability to foster critical thinking skills. If these particular systems themselves do not appear to lend themselves to this task, then at least the potential of ITS systems in general to promote critical thinking skills should be able to be ascertained.

Five ITS systems that appear to be promising in promoting critical thinking skills are *WEST*, *WUMPUS*, *SOPHIE*, *QUEST*, and *Micro-SEARCH*. The first three of the five systems are older yet classic ITS systems, whereas the last two are much newer. *Micro-SEARCH* is available for use on an IBM-PC.

WEST

The *WEST* system is a 1979 adaptation of the computer game "How the West was Won," which was originally developed on the PLATO educational system at the University of Illinois. The purpose of the original game was to assist elementary school students with basic arithmetic skills. The object of the game is to be the first one to the end of the trail by arithmetically combining three random numbers to

determine the number of spaces one can move one's token. In addition to producing the largest number, other strategies included trying to land on a particular space in order to take a shortcut and landing on an opponent's token to force him or her to move backwards. The result is that producing the largest number is not always the best strategy.

The improvement over the original game is the addition of an artificially intelligent computer coach that can offer advice if it appears that the player of the game is missing various opportunities to improve his or her performance. However, "just because a student does not use a certain skill while playing a game, one cannot conclude that he does not know that skill" (Barr, 1984, page 256). The method used to determine whether the student knows the desired skill is based upon a model that differentiates between a student's action and what the computer expert would do in a same situation. The differences between the student's action and computer expert then suggest a hypothesis about what the student does not know or has not yet mastered.

One of the important considerations in the design of the coach is to determine when it is appropriate to offer advice and how often to offer advice to the player. If the coach offers advice too often, "there is a danger that the student will never develop the necessary skill for examining his own behavior and identifying the causes of his mistakes" (Barr, 1982, page 254). On the other hand, if the coach does not offer enough advice or offers it at an inappropriate time, the student may miss opportunities to improve his or her performance. In either case, the student may lose interest in the game and not gain an increase in arithmetic skills that could be derived from the game.

Although the *WEST* system does not address critical thinking as described by Presseisen or Beyer, it does seem to promote some of the other thinking skills that both authors have discussed. For example, in addition to the obvious increase in arithmetic skills, the game helps the player in Presseisen's higher order process of decision making. Although it seems that this skill could also be gained from the original game, it seems to be more actively fostered in the newer, intelligent version. For example, if a player consistently fails to choose an obvious arithmetic combination and subsequent move, the computer coach would then bring the alternative combination to the attention of the player in an attempt to facilitate more effective decision making.

WUMPUS

Another version of an artificially intelligent computer coach is the one found in the computer game *WUMPUS*, developed in 1977. The purpose of this game is to find and destroy the Wumpus in a maze of caves while avoiding various obstacles such as bats and bottomless pits. Prior to each move into any number of adjoining caves, the system gives the player various warnings about what might be found in the caves. The player then uses logic, probability, decision theory, and geometry to find the Wumpus and avoid all of the obstacles.

As with the previous system, there is an intelligent computer-based coach called WUMPUS to assist the player. As a player reacts to the various warnings and makes various moves, an internal student model is updated. Should the student explicitly or

manifestly exhibit a particular skill, then an estimate that the student understands the skill is increased in the student model. Likewise, if a student does not show any evidence of a certain skill for a length of time, the estimate is decreased. Once the estimate that a student does not possess or understand a particular skill is sufficiently small, the coach informs the player of the various possibilities. Again, the coach must not offer advice too frequently or infrequently because the player could lose interest in the game.

Again as with the previous system, the skills learned may not directly apply to critical thinking. However, they do appear to be a part of the many micro-thinking skills discussed by Beyer. For example, the student must use logic and probability to determine whether he or she ought to move into a particular cave, given the various warnings provided by the system. The use of logic and probability are a part of the logical micro-thinking skills discussed by Beyer, where these micro-thinking skills are part of the fundamental skills required for higher order processes such as critical thinking.

SOPHIE

The *SOPHIE* system was originally developed in 1976 and is an intelligent tutoring system intended to promote problem solving skills. This is accomplished by the system answering various questions asked by users. The primary purpose is to determine whether the questions asked are logically consistent with answers already received by the student from previous questions and whether or not the student's current question is redundant.

It is interesting to note that the system does not maintain the same internal model as the user of the system might construct externally. Further, the system does not have any previously stored answers to any of the potential questions. Instead, the system must mathematically simulate the electronic processes each time it is asked a question in order to determine the correct answer. The user of the system is unaware of the internal simulation process. Although this may seem unusual, it must be remembered that it is not necessarily the answer that is important nor how the the answer is derived, but rather whether or not the questions being asked by the user are logically consistent.

Even though the problem solving skills promoted by the system are not really the same as critical thinking, problem solving is, according to Presseisen, a higher order cognitive process, as is critical thinking. It should also be noted that the type of problem solving in the *SOPHIE* system appears to be somewhat different than the type described by Presseisen and Beyer. It seems to be more involved in assessing the users' ability to ask the right types of questions, where good questioning techniques appear to be a fundamental part of both problem solving and critical thinking.

Although problem solving is ranked higher than critical thinking by some people, such as Beyer, this question is beyond the scope of this paper, and so not an issue here. The important thing to note is that this system appears to help foster higher order thinking processes.

QUEST

The *QUEST* system, developed in 1985, is similar to the *SOPHIE* system in that it helps students in the process of electronic trouble shooting. Like the *SOPHIE* system, the *QUEST* system also maintains a internal student model based upon the actions taken by the student. Although the system appears to be similar to the *SOPHIE* system, it should be noted that it is not derived from the *SOPHIE* system.

In addition to the system attempting to infer the reasons for a student's actions in order to maintain an internal student model, the *QUEST* system also receives information explicitly from the student. Unlike the *SOPHIE* system, the *QUEST* system requests that the student state his or her reasons for performing a particular action prior to actually performing their action.

At first glance, this may seem like a step backwards. It would appear as though a more sophisticated system ought to be able to infer the reasons from the action itself rather than having the user state explicitly a reason for each particular action. Although this may be a drawback given some of the earlier systems such as the *SOPHIE* system, it may be that the earlier systems did not perform as effectively as they might have under all circumstances. According to the developers "it is at the very least, extremely difficult for an *ICAI* system based on present AI methods to infer the student's mental states from his surface behaviors" (Feurzeig, 1988, page 440).

Aside from the above potential discrepancy, there is an advantage in requiring a student to state a reason prior to performing a given action. This requirement forces the student to think and then act, as opposed to just reacting to a given set of circumstances. In this sense, the *QUEST* system helps to foster some of the various aspects of critical thinking. For example, it would seem to help in identifying unstated assumptions in one's own thinking. According to the developers, the purpose of the system is "to teach conceptual sound qualitative reasoning" (page 437) and help to "shape a student's thinking" (page 439).

Micro-SEARCH

The *Micro-SEARCH* system developed in mid-1980s is an IBM PC adaptation of the *TSEARCH* system, which was originally implemented on a mainframe computer. The *Micro-SEARCH* system is included here because it is available for microcomputers and is relatively inexpensive.

The purpose of *Micro-SEARCH* is to assist students in using non-deterministic algorithms in algebra, Boolean algebra, and trigonometry. A non-deterministic algorithm is one where "there is no one correct transformation to be applied at any one stage" (Sleeman, 1987, page 70). In other words, "at any one stage, it may not be possible to decide uniquely on a single operation to apply" and one is "required to make arbitrary choices" (page 69).

For example, in solving an algebra problem such as $V = (WX + WY) / Z$ in terms of W, one could first either multiply both sides by Z or factor W out of the numerator. Then one could perform the alternative that was not chosen first. It seems that it is often the case that "the teacher frequently states the next transformation to be applied

without explaining why this is so" (page 72). The result is that a student may just memorize the method presented without exploring the other possible methods of deriving the solution.

In order to help the student see why a particular choice could be made, the *Micro-SEARCH* system encourages the student to explore all the possibilities in solving a non-deterministic problem. This is accomplished by showing the student all the possible alternatives at each stage of solving the equation along with the initial, current, and final stage of the equation. This method further allows the student to explore many of the blind alleys involved in solving an equation to help the student to discover ineffective means of problem solving. Exploring various possible methods to find a solution will, it is hoped, encourage students to reflect on their own solution process and, hence, refine their problem solving strategies.

Of all the systems reviewed in this paper, the *Micro-SEARCH* system appears to be the least "intelligent." The system does not coach the student as in the *WEST* or *WUMPUS* systems, nor does it develop an internal student model as in the *SOPHIE* or *QUEST* systems. On the other hand, the *Micro-SEARCH* system does not have a single, predefined method of deriving solutions, as might be found in non-intelligent tutoring systems; rather, the system itself derives all the various possible alternatives to the solution to an equation. Using this method, the system can assist each individual student and can be classified as an intelligent tutoring system.

In spite of the above limitations, the *Micro-SEARCH* system does appear to develop some aspects of critical thinking, where the student has the opportunity to see and evaluate the various advantages and disadvantages of pursuing various paths in solving a mathematical problem. In particular, the system appears to assist in the ability to recognize inconsistencies of particular methods of deriving solutions within the confines of mathematics and related areas.

If the system does not directly support critical thinking, it does seem to support some of the other aspects of higher order thinking processes. As pointed out by Presseisen, Lauren B. Resnick claims that one of the key features of higher order thinking is that it "is non-algorithmic. That is, the path of action is not fully specified in advance" (Pressiesen, 1986, page 10). In this sense, the *Micro-SEARCH* system appears to support at least a part of the higher order cognitive processes.

Again, although the system is not as powerful as many of the previous systems, its main advantage is that it can be run on a microcomputer. As a result, it gives us a glimpse of what might be accomplished with today's current microcomputer technology.

Conclusions

At present, the current crop of intelligent tutoring systems does not appear to foster critical thinking as defined by many of the various authors. However, these systems do seem to promote a variety of essential cognitive processes and micro-thinking skills, as well as some of the general aspects of higher order thinking processes. Further, if one looks at each of these thinking skills as a part of critical thinking, then these systems could be said to promote critical thinking indirectly.

However, strictly speaking, the current systems do not appear to foster critical thinking directly.

Although the systems of the very near future will probably not directly address critical thinking, they will most likely emphasize many of the sub-skills that form parts of the foundation upon which critical thinking is built. As the AI programmers design the software for these systems of the near future, they will also be developing the various programming skills necessary for the future ITS systems that will promote critical thinking skills. Also, as the power of computers increases, many of the algorithms necessary for these types of future, systems will be able to be implemented.

If the systems of today and the very near future do not directly promote critical thinking, how might a system of the future foster critical thinking? Such a system might present an issue to the student for him/her to ponder and judge. If necessary, the student could ask the system for further clarification. Then the student could respond to the issue and support his/her decision with various reasons for their choice. The system could then extract the fundamental arguments from the student's response, and, if necessary, ask for further clarification or sub-arguments. Upon analyzing the student's response, the system could present instances where the student's argument is weak or incomplete and also present alternative solutions or arguments to the student's response. If appropriate, the system could also provide additional arguments in support of the student's line of reasoning or thinking.

The important aspect of such systems is that there would not be a single correct answer to the issue posed by the system. What would be important is the method of thinking the student used to reach a solution. The system would not have a predefined set of answers, but rather an extensive knowledge base and reasoning base upon which to draw, given the line of reasoning or thinking the student pursued. By not having a predefined answer, any number of possible solutions could be the end result, depending on the individual student utilizing the system.

In addition to computers with more speed, more memory, and lower costs that would make such systems feasible, advances also need to be made in the area of artificial intelligence in general and natural language processing in particular. In other words, the systems would need to be able to understand the English language sentences entered by the students. This appears to be a very important capability in order to extract the elements involving the judging of various issues.

Further, as the power of microcomputer systems continues to increase and, at the same time, the price for these microcomputer systems continue to decrease, the possibility of these intelligent tutoring systems to be placed in the classroom will also increase. Again, it would seem that if any form of computer assisted instruction would have a greater chance of fostering critical thinking in the classroom, it would probably be in the area of intelligent tutoring systems implemented on microcomputers. Although at present these types of systems are still in their infancy, they are the ones that appear to have the most potential for future success.

Acknowledgement: This paper is a result of a doctoral independent study in the fall of 1989 directed by Dr. Carlton H. Bowyer, Chairman and Professor of Foundations of Education at Memphis State University.

References

Barr, A., & Feigenbaum, E. A. (1982). *The handbook of artificial intelligence* (Vol. 2). Addison-Wesley.

Beyer, B. K. (1987). *Practical strategies for the teaching of thinking*. Allyn and Bacon.

Beyer, B. K. (1988). *Developing a thinking skills program*. Allyn and Bacon.

Feurzeig, W. and Ritter F. (1988). Understanding reflective problem solving. In J. Psotka, L. D. Massey, & S. A. Mutter (Eds.), *Intelligent tutoring systems: Lessons learned*. Lawrence Erlbaum Associates.

Heiman, M. & Slomianko, J. (1985). *Critical thinking skills*. National Education Association Professional Library.

Kearsley, G. (1987). *Artificial intelligence and instruction*. Addison-Wesley.

Kearsley, G. (1988). Authoring systems for intelligent tutoring systems on personal computers. In D. H. Jonassen, *Instructional designs for microcomputer courseware*. Lawrence Erlbaum Associates.

Lawler, R. W. & Yazdani, M. (1987). *Artificial intelligence and education, Vol. 1*. Albex Publishing.

Meyers, C. (1988). *Teaching students to think critically*. Jossey-Bass.

Polson, M. C. and Richardson, J. J. (1988). *Foundations of intelligent tutoring systems*. Lawrence Erlbaum Associates.

Presseisen, B. Z. (1986). *Thinking skills: Research and practice*. National Education Society Professional Library.

Psotka, J., Massey, L. D., and Mutter, S. A. (1988). *Intelligent tutoring systems: Lessons learned*. Lawrence Erlbaum Associates.

Sleeman, D. (1987). Micro-SEARCH: A 'shell' for building systems to help students solve nondeterministic tasks. In Kearsley, G., *Artificial intelligence and instruction*. Addison-Wesley.

Critical Thinking and Open Courseware

Eduardo Rivera
InterAmerican University of Puerto Rico

Educardo Rivera is a professor in Department of Mathematics and Computer Science at InterAmerican University of Puerto Rico. His research interests include open courseware and critical thinking.

Introduction

Courseware includes computer instructional software and related material (manuals, etc.) used for instruction with a computer system. This kind of instruction, in any academic area, is called Computer Assisted Instruction (CAI). CAI courseware now consists mainly of either drill and practice or tutorial modes, with less emphasis on simulation, instructional games, and discovery modes. CAI is complemented by other non-courseware academic modes such as programming and production tools (word processing, database, graphics, or spreadsheets). This paper is concerned only with tutorial and drill and practice courseware.

Courseware has now been utilized in educational settings for more than 30 years. Practically all high schools and most elementary schools in the United States have computers and have used or still use courseware. Some have even predicted that CAI would not form an important part of school programs. Nevertheless, courseware now represents more than 50% of computer utilization time in elementary schools and 16% in high schools (Becker, 1987).

These times, however, are still times of change for computer utilization in the classroom. Most teachers are now computer literate, and many of them use computers regularly in their daily tasks. A few teachers, however, are still afraid of using computers. Computer tools are widespread in education and they will remain so; but many doubts have been raised about courseware usefulness and quality (Siegel, 1986).

Traditional courseware tends to be rapidly replaced if it does not take into account new approaches that make it a valuable tool for students and teachers. Two ideas are proposed: to design open courseware (independent of machines and open for easy adaptation by teachers) and to provide tools to design courseware that promotes critical thinking (generating open questions, problems and situations, and interactions to analyze answers).

Assessment of Traditional Software

"The belief that computers can provide benefits and the optimistic expectation that they are supplying these benefits have fueled a tremendous growth in the use of computers in schools over the past several years. But in the long run educators will lose faith in the value of computer-based instruction if objective evaluations begin to accumulate evidence that computers have made little difference in the education of most children and adolescents" (Becker 1987, page 162).

As Becker suggested, the educational community is disenchanted with courseware and computer uses in learning. This technology has not achieved all the anticipated levels of use. Although computers can provide interactive, fast, friendly 'multimedia courses, many students chose not to continue with CAI after they have completed one programmed course. They feel that courseware is slow, repetitive, and boring.

One common belief is that the boredom from educational software comes from the student's assumption that computer programs should provide entertainment as in arcades or Nintedo-like games. While entertainment is not a primary goal of courseware, it is true that courseware needs imagination (Siegel, 1986) in order to capture student attention. Others believe that courseware is often simplistic and does not offer sufficient graphics (Naiman, 1987). Nevertheless, sophistication is not always better than a simple program with clear objectives.

At the beginning, teachers are usually very enthusiastic about the computing courseware approach. But after some time, regardless of the results, they abandon it. Either they perceive computers as another rigid instructional technology (such as audiovisuals), or they feel that the courseware does away with the teacher in the instructional process. Perhaps because there is an increasing shortage of teachers in mathematics and science, one of the main trends in the development of courseware has been to follow the common track of many educational technologies: to substitute the technology for the teacher (Bork, 1987).

Another problem contributing to the lack of good quality software is that the market is driven by educational publishers who cannot afford to take risks with innovative courseware. Because of the uniformity of mass market courseware, it is cheaper to produce, whether or not it is adequate for all of the masses. Regardless of reasons, the teacher is most often forced to accept what is offered in the marketplace.

Finally, computer-based instruction is still viewed as a last minute addendum to the ongoing syllabus (Naiman, 1987) rather than as a unified part of a course. In order to use courseware effectively in a classroom, it must be a continuation of the teacher's instructional method. If courseware provided such opportunities, considerable change or adaptation of the curriculum could take place.

We suggest here that the main reason classroom courseware has failed is that it comes as a closed package that cannot be modified, merged with other packages, or adapted by the teacher. Teachers do not have access to the code or logic of the program, so even if they were programmers, they cannot modify a single word of an instructional lesson. Teachers feel that they are themselves used by the courseware and that the teacher's role is only to play the software. The teacher is put into a passive role. Hence, teachers have little enthusiasm for the courseware, and this lack of enthusiasm is transmitted to students. The result is that courseware is used from time to time as merely another audiovisual tool and not as a privileged resource for learning, as it was once expected (Lorimy, 1985).

Authoring Courseware

One obvious solution to the above problems is the scenario where every teacher authors his/her own courseware. Many questions have been raised about the possibility of having the teacher as an author. In fact, many authoring tools have been created by vendors for the teacher to use, but with very little implementation success.

Many authoring languages have been designed, with PILOT perhaps being the best known (Merrill, 1987). These languages have now evolved to Courseware Authoring Tools (CAT) or Authoring Systems that provide a framework for lesson development. These kinds of tools provide author aids that are easy to use and with many attractive features such as sound, music, color, high resolution images, or animation. However, the writing of quality courseware is more difficult than writing a book. (One needs to be concerned not only about the content of the courseware, the multimedia, interaction, sequences, etc.)

Perhaps the most difficult problem facing a teacher who wishes to become a courseware author is the availability of time. It is estimated that for each hour of computer-assisted instruction marketed by a commercial educational software house, development has taken up to 200 hours. Teachers simply do not have such time to devote to the creation of materials.

It is evident that teachers must be educated to evaluate and utilize commercially available courseware. Such courseware would be of great value if the teacher could more easily adapt it to their lesson plans.

Open Courseware

Three main hypothesis are behind the trend of closed courseware, i.e. software that may not be modified. First, most traditional courseware tries to be a substitute for the teacher. Such courseware can be used independently of the teacher. Second, teachers are not capable (because of time constraints or knowledge) of creating good quality tutorials, courseware, or other computer-related material. Thus, vendors must provide the bulk of CAI materials. Finally, the consumer power of teachers can be manipulated in the sense that they continue to consume mass-produced courseware, which is cheaper to produce as an industrialized product.

With this in mind, we recommend creation of open courseware. Such courseware is:

- Portable and independent or quasi-independent of a computer trademark or architecture.
- Modular so that it can be cut and pasted with other modules of other producers or with modules the teacher produces to arrive at a complete lesson.
- Adaptable by the teacher to the particular background and needs of students. (These modifications do not necessarily require the rewriting of programming code, but may be made through tool support.)
- Easy to incorporate tools or other programs to facilitate student interaction.
- Free of an imposed sequence of learning.
- Bidirectional, offering the possibility that the student can ask for information or an explanation.

- Fosters critical analysis, i.e., provides tools or guides to open questions and critical thinking exercises (not only mechanical exercises or quizzes).
- Interactive, i.e., provides a medium to interact with other students or teachers.
- Teacher oriented, providing a tool for uses in the classroom.

Teachers using open courseware would be actors and participants, and not just spectators of the learning process. This would be possible without complex programming. Teachers must:

- Evaluate and select material and courseware.
- Change the material proposed or suggested by the courseware, if necessary.
- Incorporate new instructional material into the courseware.
- Provide guides and tools to assist the students using the courseware.
- Generate and select intelligent questions.
- Review open answers such as those from critical thinking. (This review of answers would still be a "manual" task, perhaps assisted by the computer. Such revision would stimulate discussion and require human interaction.)

The production of open courseware would renew and rejuvenate teacher enthusiasm in the use of CAI that has, of late, been fading.

Critical Thinking and Open Courseware

Because education is concerned with understanding and not merely information transfer (Kemp cited by Meeks, 1987), the mechanical or rigid scheme of question and answer do not assure that the student has mastered a subject. In 1981, Simon in his report to the president of France about education and computing, says that CAI gives only directives and does not provide sufficient stimulus to the students over a period of instruction. The critical spirit is not developed among students with traditional courseware. Any proposition of open courseware must follow the basic ideas of critical thinking.

We think that open courseware as defined above should be used for working in the classroom. However, the design of such software requires a strong background in critical thinking and problem solving skills on the part of the author and the teacher. Open courseware can present the possibility of discussing and learning interactively: student-machine, student-teacher, and student-student. This bidirectionality is inherent to improved critical thinking.

One of the most pressing difficulties of courseware design is to have programmed all possible answers. But with open courseware, this requirement no longer exists as the teacher continues to monitor answers. Critical thinking analysis must be an inherent part of this answer monitoring.

Conclusions

Because of the shortcomings associated with closed courseware, "the long term effect of the market vacuum may be to abort educational computing before it has the chance to develop its full potential" (Siegel, 1986). It would be a pity if this generation of teachers and software producers abandon an opportunity to improve education .

Schools are an economic consumer force. They must require designers, software houses, publishers, and computer companies to build a new type of courseware. Enthusiasm for using computers should be increased and, most importantly, through adaptation and the use of critical thinking techniques, the quality of education will be enhanced.

Acknowledgement

The author recognizes Dr. Wayne Wostenholme who carefully reviewed and provided comments on this work.

References

Becker, H. (1987). Using computers for instruction. *Byte, 12*(2), 149-162.

Bork A. (1987). The potential for interactive technology. *Byte, 12*(2), 201-206.

Lorimy B. (1985). Agence de l'informatique, *Rapport E.A.O.* ("Report C.A.I." in French).

Meeks B. (1987). The quiet revolution, on-line education becomes a real alternative. *Byte 12*(2), 183-190.

Merrill, P., Tolman, M., Christensen, L., Hammons, K., Vincent, B., & Reynods, P. (1987). *Computers in education*. Englewood Cliffs, NJ: Prentice Hall, .

Naiman A. (1987). A hard look at educational software. *Byte 12*(2), 193-200.

Siegel, M. & Davis, D. (1986). *Understanding computer based education*. New York: Random House.

"What Can We Learn from Each Other's Experiences?": Observations of a Research-Oriented Workshop by a Classroom Teacher

Sylvia Robinson, Winston-Salem, North Carolina

Sylvia Robinson is a classroom teacher in Winston-Salem, North Carolina. Her research interests include restructuring classroom teaching and using computers in the classrooms.

Activities of the Day

As coordinators for this research workshop, Problem Solving and Critical Thinking, Drs. Jim Poirot and Cathie Norris from the Department of Computer Education and Cognitive Systems of the University of North Texas, welcomed the fourteen participants.

Their professional backgrounds were as varied as their ages, estimated to range from 28 to 55. Work settings of participators included full-time graduate study, department chair of an undergraduate teacher training programs, nursing educator, NASA engineering software designer, and university professors of computer science. But all were concerned with problem solving and critical thinking in educational settings enriched by computer technology.

Presenters had been chosen on the basis of a submitted abstract of a position paper related to the following research questions:

> What is "critical thinking"; how does one measure critical thinking and problem solving abilities; how can technology be used in teaching problem and critical thinking; can theory support classroom practice; do problem solving skills in one discipline transfer to skills in another discipline; what strategies do we use in solving certain classes of problems; does one's socioeconomic status or sex affect one's critical thinking or problem solving skills; do computer programming abilities transfer to problem solving abilities in other disciplines?

Dr. Norris focused the group's thinking with seven foundation questions related to the problem solving issues. Dialogue began and lively interchanges occurred about the definitions of problem solving and critical thinking. General agreement on defining a problem as "a situation with a blocked goal . . . an impasse" was more easily reached than a consensus on the definition of problem solving. One participant maintained that all human behavior was goal directed and emphasized the critical importance of language and culture. This was gently challenged but accepted, and the

participants further agreed that problem solving is not the same as critical thinking. They agreed to disagree about the ways these thought processes differ, which set the stage for the sharing of their current research findings.

Although the epistomological and pedagogical concepts of the group members varied considerably, there was no real disputing the notion that the learning environment of computer technology uniquely supports the development of problem solving abilities. Most importantly and conclusively, however, was the general agreement [corroborated by Swan's work in Albany, Moseley's in Huntsville, Harris' in Dallas, and Dunn's in Long Island] that the higher level thinking skills must be explicitly taught, consciously generalized and internalized, and then intentionally practiced and monitored before students consistently transfer these strategies to other domains. Development of critical thinking may sometimes happen through instinctive "modeling," but must never be assumed to be automatic. One participant pointed out that the predictable prescriptions for thinking that are already known are useful for valuable teaching/learning.

Observer's Comments

We believe sophisticated educational technology offers the greatest hope for improvement in schools today, because computers are powerful change agents in the classroom. However, they are tools or vehicles for critical thinking and problem solving and only indirectly nurture these human processes. The use of technology can fast forward learning in this information age because interacting with a computer in a cooperative environment empowers students to assume more ownership. It is this increased responsibility for learning that encourages the self-motivation and human persistence that produces superior thought processing and more critical judgments. Technology can encourage students to do more meaningful activities and relate their learning more fully to the real world. Learning happens when behavior changes as a result of interacting with people and the environment. Computer technology simulates learning environments that are difficult to create and can fertilize traditional curricula content with real world knowledge. Nevertheless, the whole point of education is to nurture the ability to form independent, valid judgments, not the collecting of intellectual lint. Far too little research is being done on the acquisition of critical thinking and problems solving strategies, the underpinnings of all else.

Schools currently are attempting to function with antiquated feedback systems that are inefficient and ineffective. Precious resources are limited; educational achievement is dismal; and our children's needs are largely unmet. There must be dynamic interaction among those on the cutting edge of educational technology, the best minds in pedagogy, and school decisionmakers. In the aftermath of the removal of the Berlin Wall, there must be renewed effort to tear down the barriers that have traditionally separated basic educational research from actual teaching practices in the public school classrooms. Insightful research from the school room teaching of critical thinking and problem solving strategies could lead to more strength sharings. The sense of responsibility, too much in default, and the eroded commitment to prepare our

children to face the future can be revitalized through the continued support and expansion of research workshops like this.

Therefore, as public school classroom teachers, we are delighted and sustained by the interest and work of the researchers in these foundational issues, which are often lost in the commercially hyped attention to bells and whistles technology. Substance and value are the content, for the essence of critical thinking is proper evaluation. There can be critical thinking in the use of imagery in multimedia, and it may be more useful to our children's future problem solving needs than the linear processes incorporated so heavily in current CAI. The reciprocal influences of technology and thinking are dramatic and powerful, but a wise tool maker concerned with the good of humanity better be in control.